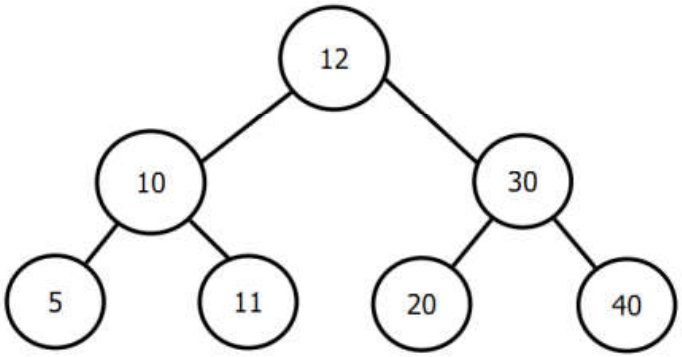
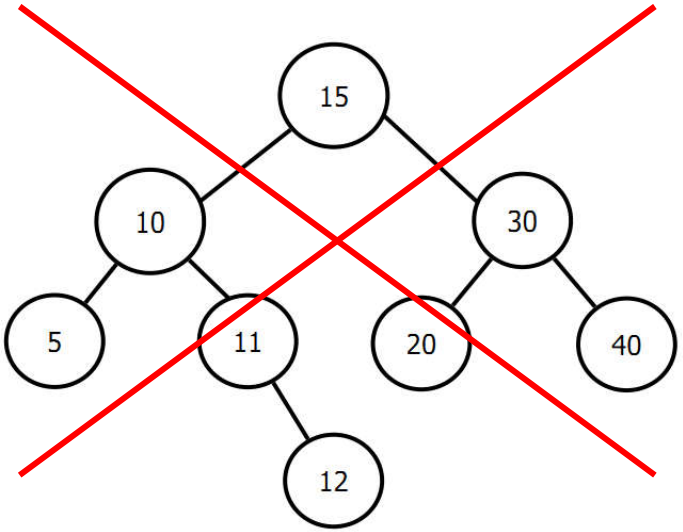
Árvores Binárias

Árvore estritamente binária

É uma árvore onde todos os nós que não são folha possuem dois filhos.

Ex.:





Árvore binária completa

Uma árvore binária completa de profundidade *d* é uma árvore estritamente binária onde todas as folhas estão no nível *d*.

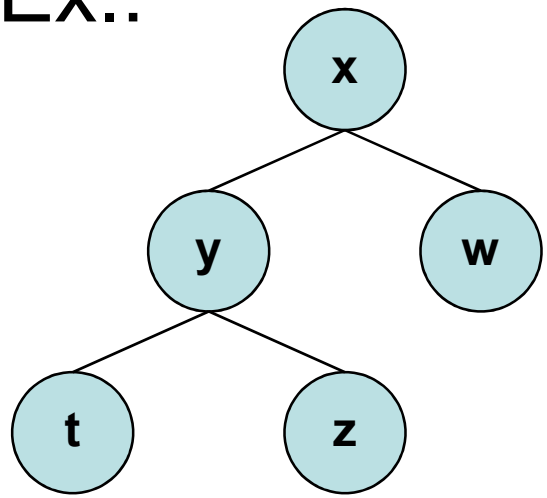
Árvores Binárias

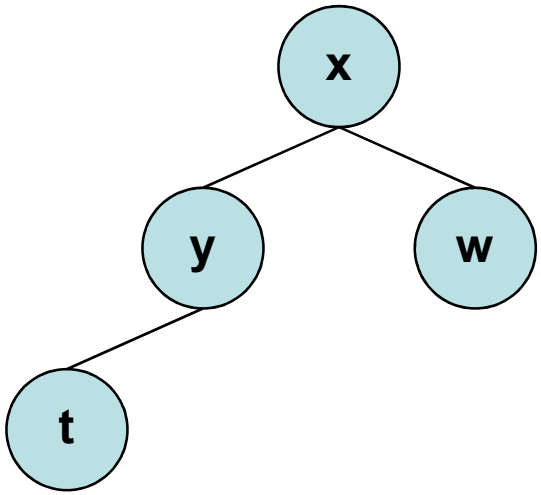
Árvore binária quase completa

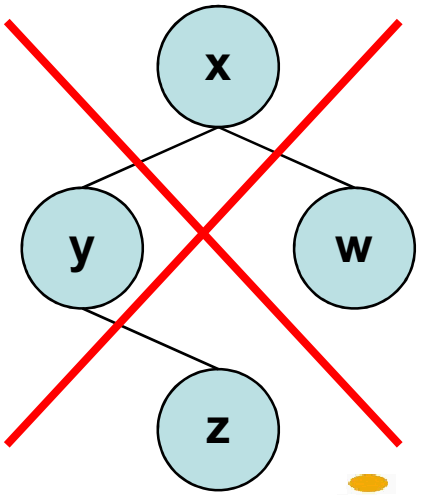
Uma árvore binária de profundidade *d* será uma árvore binária quase completa se:

1. **Cada folha da árvore estiver no nível *d* ou no nível *d*-1.**
2. **Para todo nó *nd* que possui um descendente direito no nível *d*, todo descendente esquerdo de *nd* é folha no nível *d* ou tem 2 filhos.**

Ex.:







Árvores Binárias

Assim como vimos em nosso estudo sobre listas, árvores também podem ser armazenadas de forma estática ou dinâmica.

Começaremos nosso estudo com o armazenamento estático.

Quais campos seriam relevantes para cada nó?

Os campos *info*, *left*, *right* e *father*. Onde estes campos representam respectivamente a informação armazenada no nó, seu filho à esquerda, seu filho à direita e seu pai.

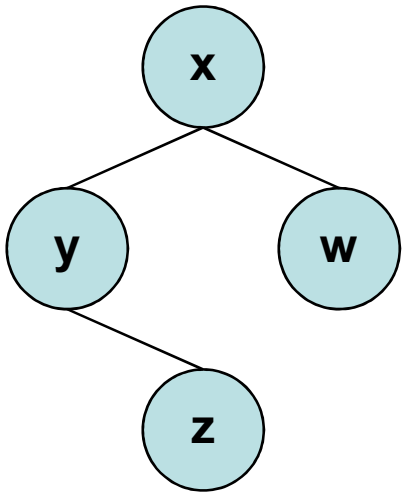
UNIVASF

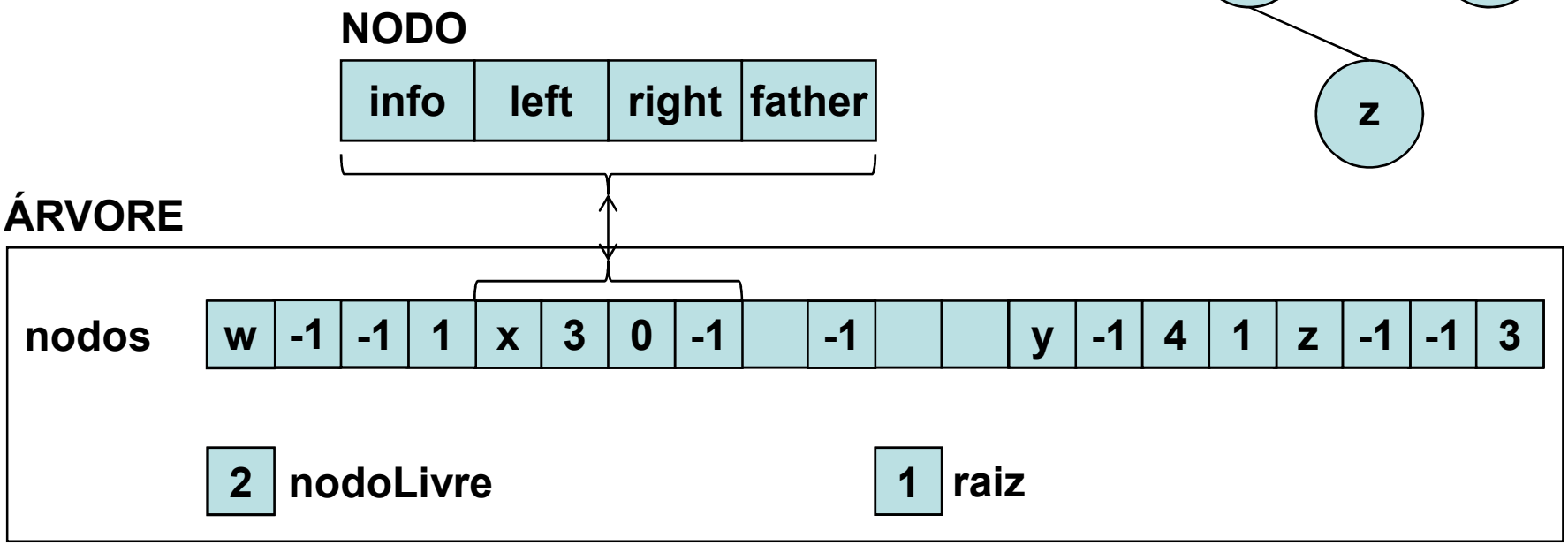
Árvores Binárias

Como vocês sugeririam o armazenamento estático?

Em um vetor?

Como?





Árvores Binárias

Quais operações primitivas seriam relevantes?

Existem várias operações primitivas que podem ser aplicadas a uma árvore binária.

Se ***p*** é uma referência para um nó ***nd*** de uma árvore binária associada a ***t***, a função ***info(t, p)*** retorna o conteúdo de ***nd***. As funções ***left(t, p)***, ***right(t, p)***, ***father(t, p)*** e ***brother(t, p)*** retornam referências para o filho esquerdo de ***nd***, filho direito de ***nd***, pai de ***nd*** e irmão de ***nd***, respectivamente. Estas funções retornaram referência inválidas se ***nd*** não tiver filho esquerdo, filho direito, pai ou irmão.

Árvores Binárias

Temos também as funções lógicas ***isleft(t, p)*** e ***isright(t, p)*** que retornaram o valor ***true*** se ***nd*** for um filho esquerdo ou direito, respectivamente, de algum outro nó na árvore associada a ***t***, e ***false***, caso contrário.

Para construir uma árvore binária, as operações ***maketree***, ***setleft*** e ***setright*** são úteis.

maketree(t, x) cria uma nova árvore associada a ***t*** consistindo num único nó com o campo de informação ***x***.

Árvores Binárias

setleft(t, p, x) recebe uma referência ***p*** para um nó da árvore binária associada a ***t*** sem filho esquerdo e cria um novo filho esquerdo para o nó referenciado por ***p*** com o campo informação ***x***.

setright(t, p, x) análoga a ***setleft***, exceto pelo fato de que ela cria um filho direito no nó referenciado por ***p***.

Com o que foi definido podemos definir, na linguagem C, uma árvore binária como sendo:

```
/*Definição a ARV_BIN_SEQ*/
#define NUMNODES 100
typedef struct {
   int info;
   int left;
   int right;
   int father;
} NODE;
typedef struct{
   int root;
   int nodeFree;
   NODE nodes[NUMNODES];
}ARV_BIN_SEQ;
void maketree(ARV_BIN_SEQ *, int);
void setleft(ARV_BIN_SEQ *, int, int);
void setright(ARV_BIN_SEQ *, int, int);
int info(ARV_BIN_SEQ *, int);
int left(ARV_BIN_SEQ *, int);
int right(ARV_BIN_SEQ *, int);
int father(ARV_BIN_SEQ *, int);
int brother(ARV_BIN_SEQ *, int);
int isleft(ARV_BIN_SEQ *, int);
int isright(ARV_BIN_SEQ *, int);
```

Árvores Binárias - Alocação Sequencial

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

```
/*Definição a ARV_BIN_SEQ*/
#define NUMNODES 100
typedef struct {
   int info;
   int left;
   int right;
   int father;
} NODE;
typedef struct{
   int root;
   int nodeFree;
   NODE nodes[NUMNODES];
}ARV_BIN_SEQ;
```

```
void maketree(ARV_BIN_SEQ *, int);
void setleft(ARV_BIN_SEQ *, int, int);
void setright(ARV_BIN_SEQ *, int, int);
int info(ARV_BIN_SEQ *, int);
int left(ARV_BIN_SEQ *, int);
int right(ARV_BIN_SEQ *, int);
int father(ARV_BIN_SEQ *, int);
int brother(ARV_BIN_SEQ *, int);
int isleft(ARV_BIN_SEQ *, int);
int isright(ARV_BIN_SEQ *, int);
```

```
void maketree(ARV_BIN_SEQ *t, int x)
{
   int i, ind;
   for (i=0; i<NUMNODES-1; i++)
      t->nodes[i].left = i+1;
   t->nodes[i].left = -1;
   t->nodeFree=0;
   ind = getNode(t);
   if (ind != -1)
   {
      t->nodes[ind].info = x;
      t->nodes[ind].left = -1;
      t->nodes[ind].right = -1;
      t->nodes[ind].father = -1;
      t->root = ind;
   }
   else
   {
      printf("Impossivel construir a arvore!");
      exit(1);
   }
}
```

```
int getNode(ARV_BIN_SEQ *t)
{
   if (t->nodeFree != -1)
   {
      int i = t->nodeFree;
      t->nodeFree = t->nodes[t->nodeFree].left;
      return i;
   }
   else
      return -1;
}


void freeNode(ARV_BIN_SEQ *t, int node)
{
   t->nodes[node].left = t->nodeFree;
   t->nodeFree = node;
}
```

UNIVASF

Árvores Binárias - Alocação Sequencial

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

```
/*Definição a ARV_BIN_SEQ*/
#define NUMNODES 100
typedef struct {
   int info;
   int left;
   int right;
   int father;
} NODE;
typedef struct{
   int root;
   int nodeFree;
   NODE nodes[NUMNODES];
}ARV_BIN_SEQ;
```

```
void maketree(ARV_BIN_SEQ *, int);
void setleft(ARV_BIN_SEQ *, int, int);
void setright(ARV_BIN_SEQ *, int, int);
int info(ARV_BIN_SEQ *, int);
int left(ARV_BIN_SEQ *, int);
int right(ARV_BIN_SEQ *, int);
int father(ARV_BIN_SEQ *, int);
int brother(ARV_BIN_SEQ *, int);
int isleft(ARV_BIN_SEQ *, int);
int isright(ARV_BIN_SEQ *, int);
```

```
void setleft(ARV_BIN_SEQ *t, int p, int x)
{
   int ind = getNode(t);
   if (ind != -1)
   {
      t->nodes[p].left = ind;
      t->nodes[ind].info = x;
      t->nodes[ind].left = -1;
      t->nodes[ind].right = -1;
      t->nodes[ind].father = p;
   }
   else
   {
      printf("Impossivel inserir filho a esquerda!");
      exit(2);
   }
}
```

Árvores Binárias - Alocação Sequencial

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

```
/*Definição a ARV_BIN_SEQ*/
#define NUMNODES 100
typedef struct {
   int info;
   int left;
   int right;
   int father;
} NODE;
typedef struct{
   int root;
   int nodeFree;
   NODE nodes[NUMNODES];
}ARV_BIN_SEQ;
```

```
void maketree(ARV_BIN_SEQ *, int);
void setleft(ARV_BIN_SEQ *, int, int);
void setright(ARV_BIN_SEQ *, int, int);
int info(ARV_BIN_SEQ *, int);
int left(ARV_BIN_SEQ *, int);
int right(ARV_BIN_SEQ *, int);
int father(ARV_BIN_SEQ *, int);
int brother(ARV_BIN_SEQ *, int);
int isleft(ARV_BIN_SEQ *, int);
int isright(ARV_BIN_SEQ *, int);
```

```
void setright(ARV_BIN_SEQ *t, int p, int x)
{
   int ind = getNode(t);
   if (ind != -1)
   {
      t->nodes[p].right = ind;
      t->nodes[ind].info = x;
      t->nodes[ind].left = -1;
      t->nodes[ind].right = -1;
      t->nodes[ind].father = p;
   }
   else
   {
      printf("Impossivel inserir filho a direita!");
      exit(2);
   }
}
```

Árvores Binárias - Alocação Sequencial

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

```
/*Definição a ARV_BIN_SEQ*/
#define NUMNODES 100
typedef struct {
   int info;
   int left;
   int right;
   int father;
} NODE;
typedef struct{
   int root;
   int nodeFree;
   NODE nodes[NUMNODES];
}ARV_BIN_SEQ;
```

```
void maketree(ARV_BIN_SEQ *, int);
void setleft(ARV_BIN_SEQ *, int, int);
void setright(ARV_BIN_SEQ *, int, int);
int info(ARV_BIN_SEQ *, int);
int left(ARV_BIN_SEQ *, int);
int right(ARV_BIN_SEQ *, int);
int father(ARV_BIN_SEQ *, int);
int brother(ARV_BIN_SEQ *, int);
int isleft(ARV_BIN_SEQ *, int);
int isright(ARV_BIN_SEQ *, int);
```

```
int info(ARV_BIN_SEQ *t, int p)
{
  return t->nodes[p].info;
}

int left(ARV_BIN_SEQ *t, int p)
{
  return t->nodes[p].left;
}

int right(ARV_BIN_SEQ *t, int p)
{
  return t->nodes[p].right;
}

int father(ARV_BIN_SEQ *t, int p)
{
  return t->nodes[p].father;
}
```

UNIVASF

Árvores Binárias - Alocação Sequencial

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

```
/*Definição a ARV_BIN_SEQ*/
#define NUMNODES 100
typedef struct {
   int info;
   int left;
   int right;
   int father;
} NODE;
typedef struct{
   int root;
   int nodeFree;
   NODE nodes[NUMNODES];
}ARV_BIN_SEQ;
```

```
void maketree(ARV_BIN_SEQ *, int);
void setleft(ARV_BIN_SEQ *, int, int);
void setright(ARV_BIN_SEQ *, int, int);
int info(ARV_BIN_SEQ *, int);
int left(ARV_BIN_SEQ *, int);
int right(ARV_BIN_SEQ *, int);
int father(ARV_BIN_SEQ *, int);
int brother(ARV_BIN_SEQ *, int);
int isleft(ARV_BIN_SEQ *, int);
int isright(ARV_BIN_SEQ *, int);
```

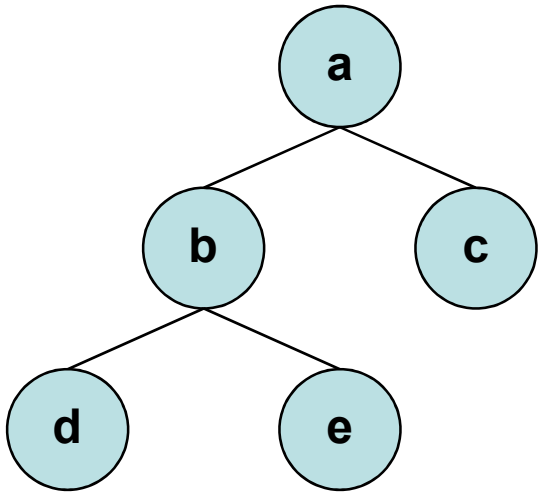
```
int brother(ARV_BIN_SEQ *t, int p) {
   if (father(t, p) != -1) /*Se não for a raiz*/
      if (isleft(t, p))
         return right(t,father(t, p));
      else
         return t->nodes[t->nodes[p].father].left;
   return -1;
}

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.

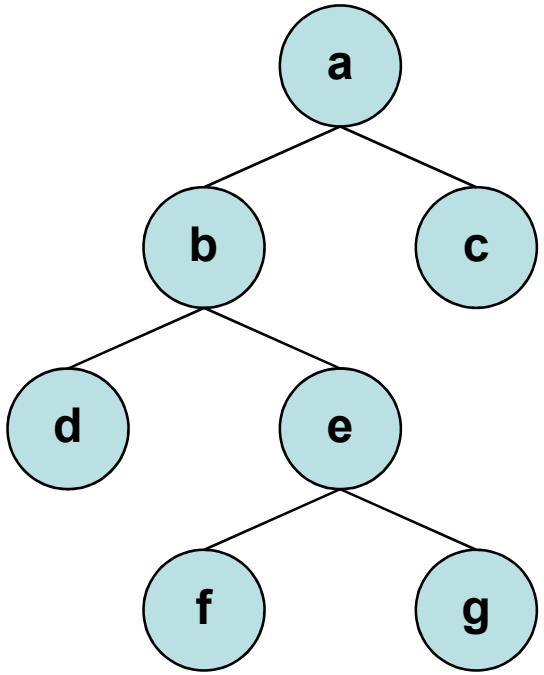


Esta Árvore poderia ser armazenada em um vetor da seguinte forma:

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

?

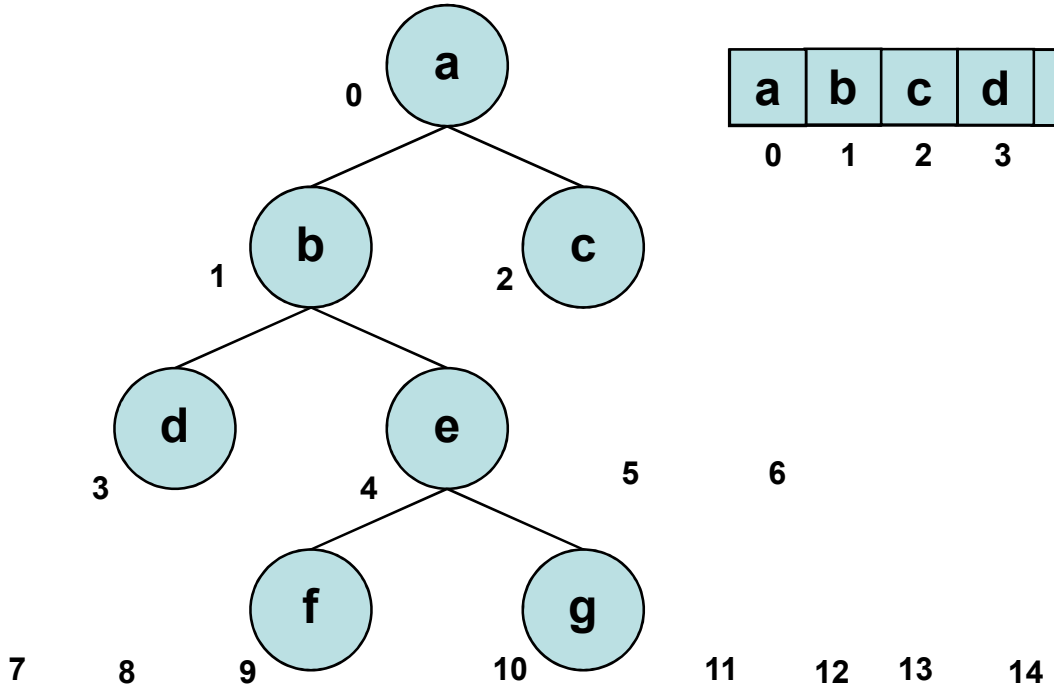
E se fosse assim:

a	b	c	d	e					f	g				

Vocês identificam alguma vantagem?

Árvores Binárias

Árvore:



Representação:

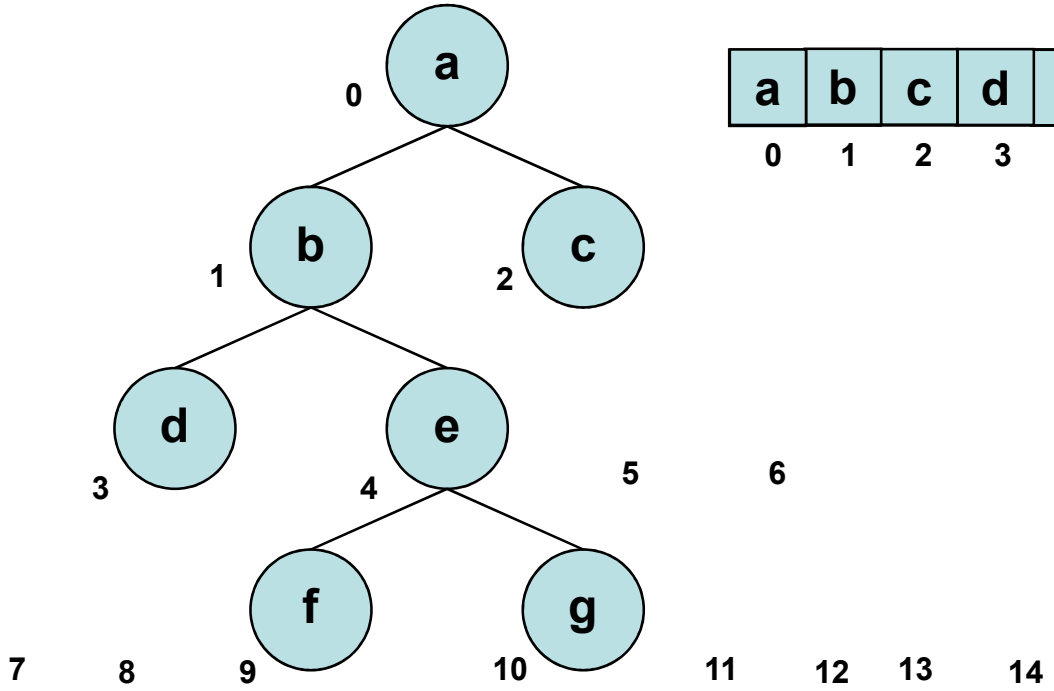
a	b	c	d	e					f	g				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

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

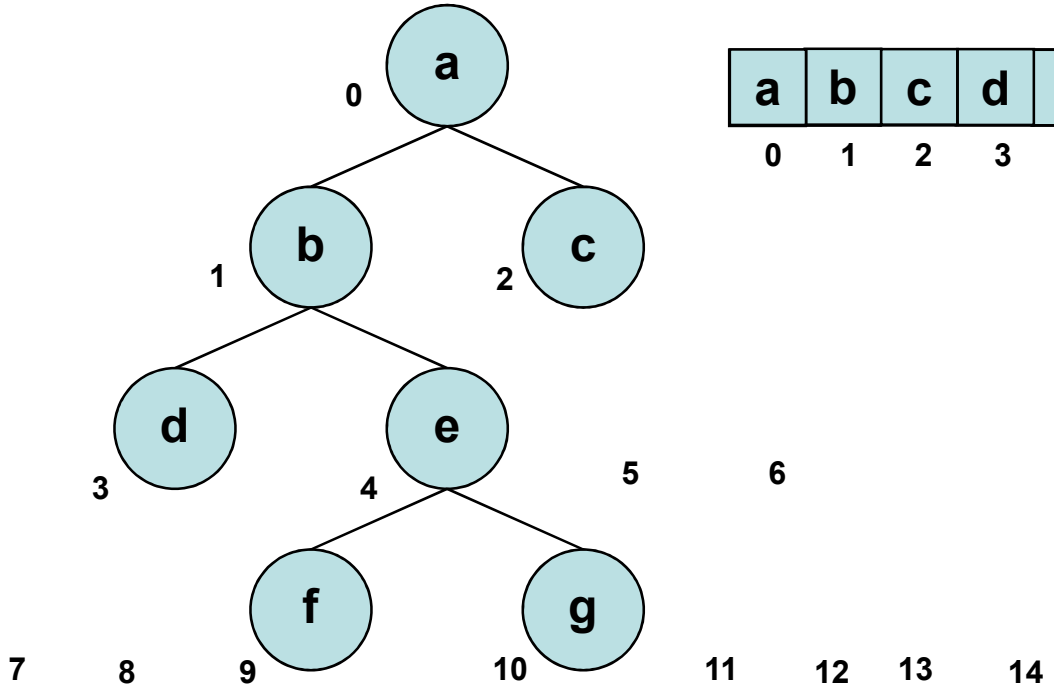
Como saber quem é o pai de um nó?

(p-1)/2.

p é índice do vetor que contém o nó.

Árvores Binárias

Árvore:



Representação:

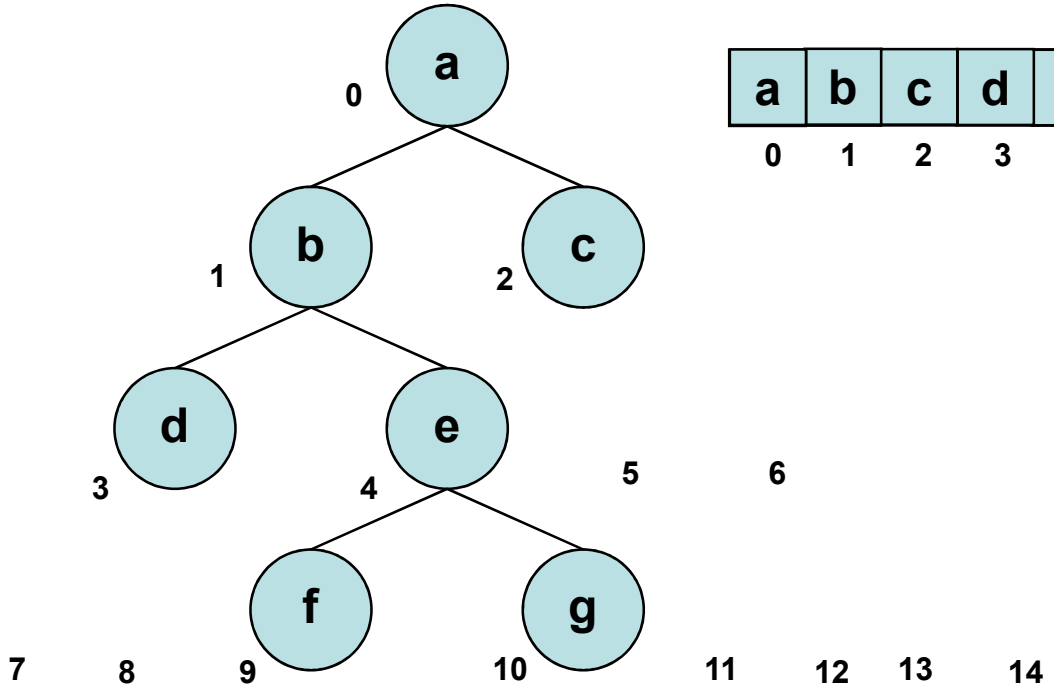
a	b	c	d	e					f	g				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Como saber quem é o filho esquerdo de um nó?

2*p+1

Árvores Binárias

Árvore:



Representação:

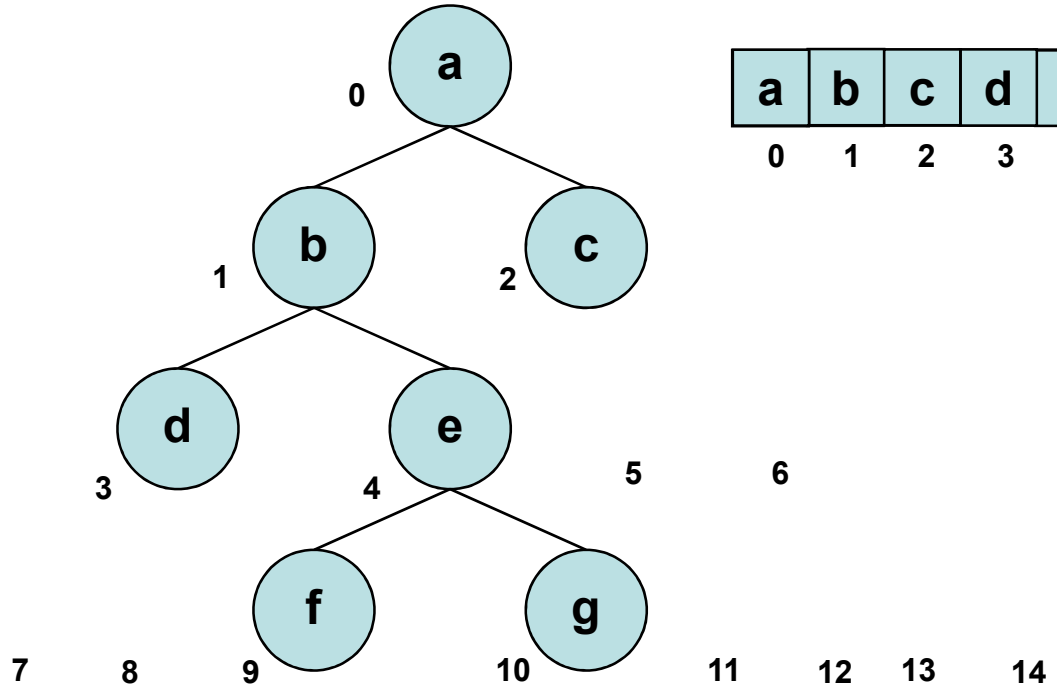
a	b	c	d	e					f	g				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

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?

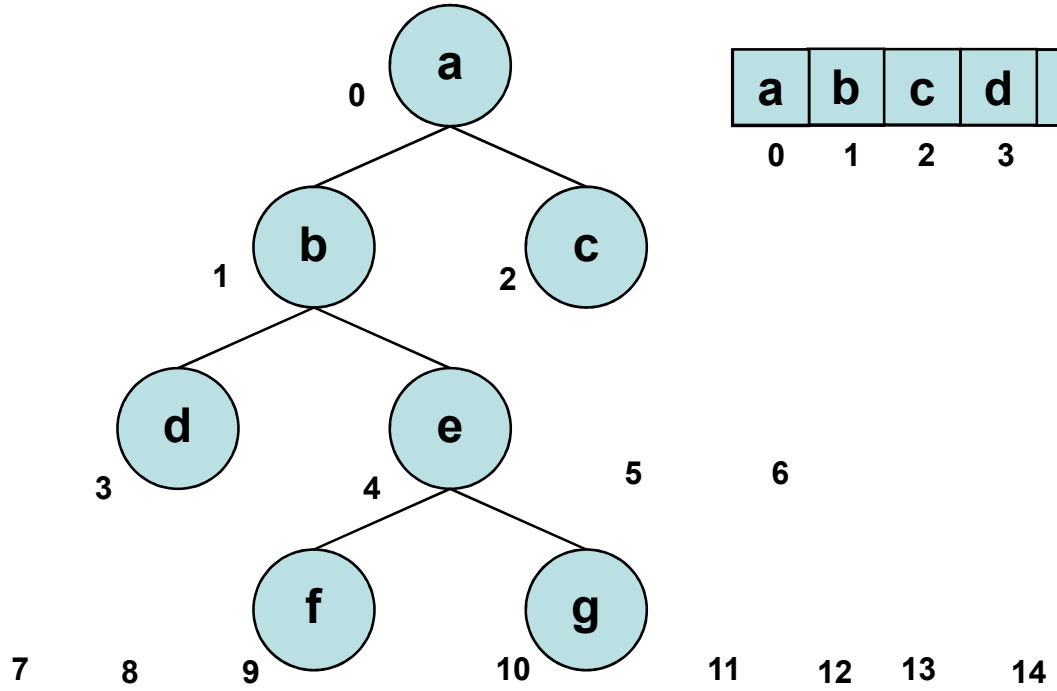
Primeiro, devemos saber se ele é um filho esquerdo.

Como saber?

É só verificar se p é impar.

Árvores Binárias

Árvore:



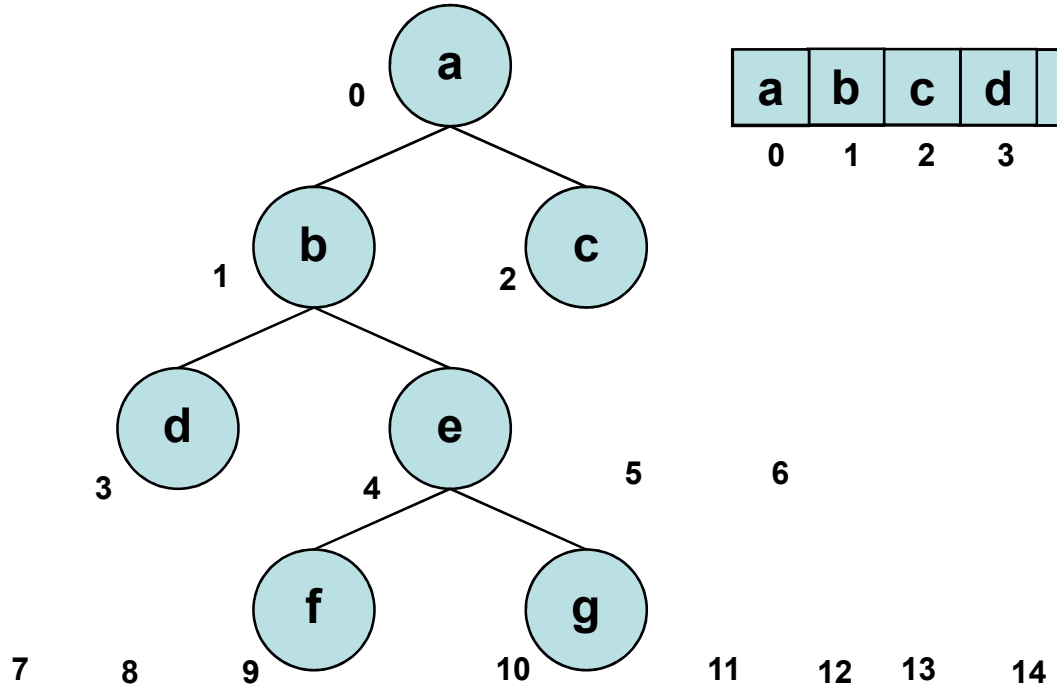
Representação:

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:



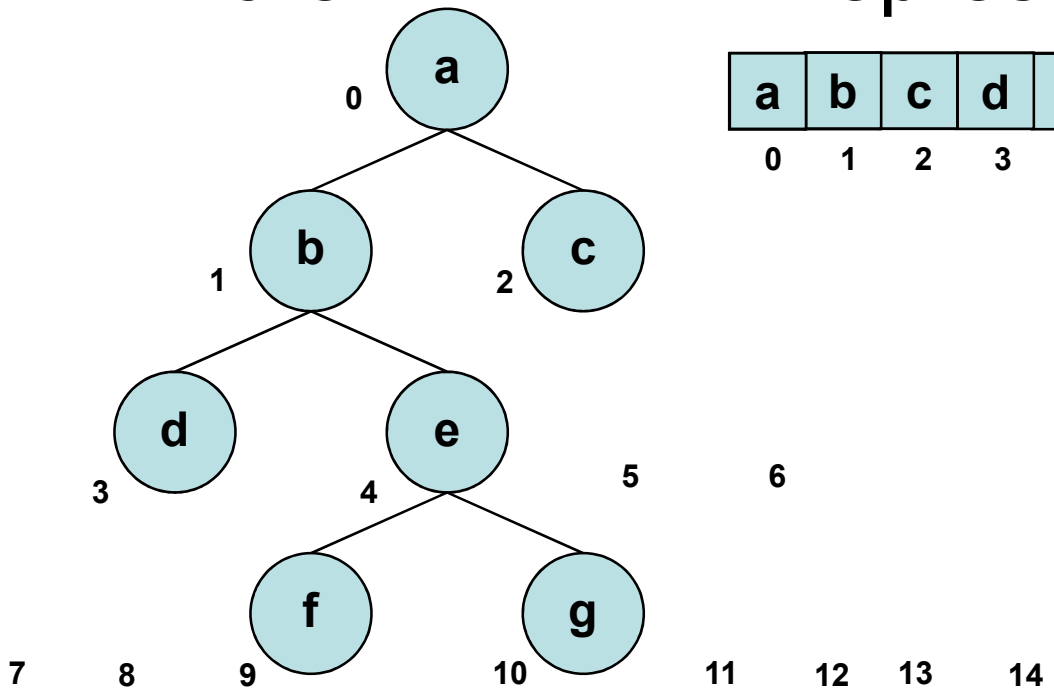
Representação:

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:

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

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);
```