Aula 05 - Conteúdo

1) Árvores e suas representações
2) Árvore Binária e suas representações
3) Percursos em Árvores Binárias
4) Implementações dos algoritmos sobre árvores binárias
5) Representação de árvores por meio de árvores binárias
6) Exercícios

Árvores

Terminologia Básica

Árvore é um conjunto finito de um ou mais nós de tal natureza que:

(i) existe um nó especialmente designado denominado raiz.

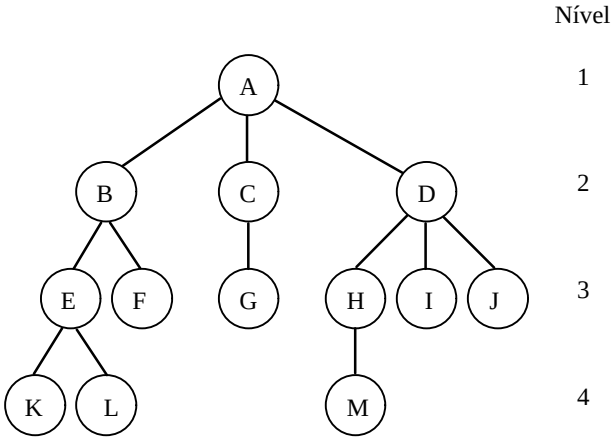
(ii) os nós restantes estão desdobrados em n conjuntos (n ≥ 0) separados $T_1, T_2 \ldots T_n$ em que cada um dos referidos conjuntos se constitui numa árvore. Os conjuntos $T_1, T_2 \ldots T_n$ se denominam as **subárvores** da raiz.

Obs.: $T_1, T_2 \ldots T_n$ são conjuntos individuais, ou seja, subárvores não podem ser interligadas (não deve haver procriação entre elementos).

Definições:

- **grau de um nó**: número de subárvores de um nó
- **grau da árvore**: grau máximo entre os nós da árvore
- **folhas**: nós com grau 0 (também chamados nós terminais)
- **filhos**: raízes das subárvores de um nó
- **pai de um nó**: raiz da subárvore que contem este nó
- **irmãos**: nós que são filhos do mesmo pai
- **nível**: define- se inicialmente que a raiz está no nível 1. A partir deste ponto temos que os filhos da raiz estão no nível 2, os filhos dos filhos da raiz estão no nível 3 e assim sucessivamente.
- **altura ou profundidade**: nível máximo entre os nós de uma árvore
- **antepassados de um nó**: seqüência de nós ao longo da raiz até este nó
- **floresta**: conjunto de n árvores separadas (n ≥ 0). A noção de floresta se aproxima muito da noção de árvore pois se retirarmos a raiz obtemos uma floresta

Exemplo 1:



Podemos afirmar sobre a árvore acima:

- a árvore tem 13 nós
- a raiz é o nó **A**
- o grau da árvore é 3
- o grau do nó **A** é 3, o grau do nó **C** é 1 e o grau do nó **F** é 0
- as folhas da árvore são {**K, L, F, G, M, I, J**}
- os demais nós são chamados de nós não terminais
- os filhos do nó **D** são os nós **H**, **I** e **J**
- o pai do nó **D** é o nó **A**
- os nós **H**, **I** e **J** são irmãos.
- os antepassados do nó **M** são os nós **A**, **D** e **H**.
- nível do nó **B** é 1, do nó **H** é 2 e do nó **M** é 3
- a altura da árvore é 4

Representação de Árvores

a) estrutura em forma de lista
A árvore do exemplo 1 pode ser representada pela seguinte lista de nós:
(A(B(E(K,L),F),C(G),D(H(M),I,J)))

b) estrutura utilizando apontadores

Dados	Apont. 1	Apont. 2	...	Apont. n

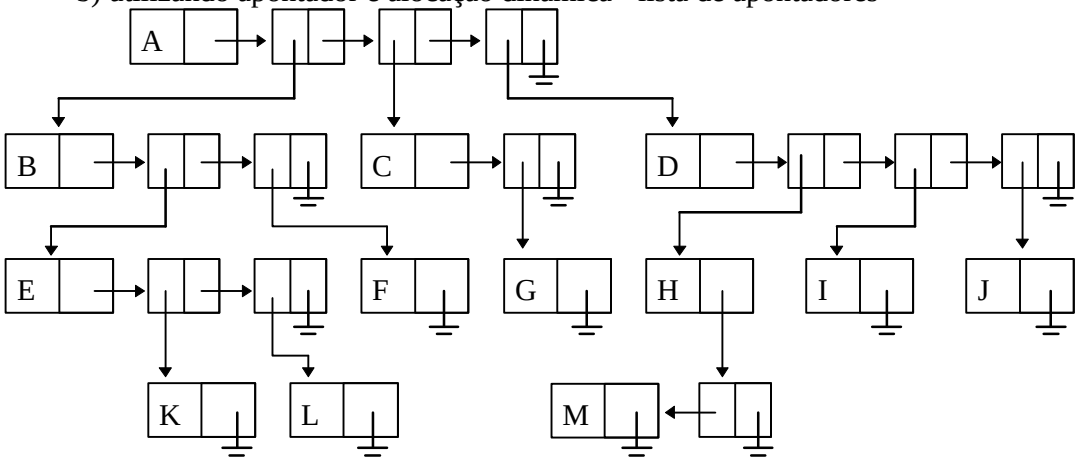
Exemplos de representação com apontadores

a) utilizando vetor de apontadores

	C	G	M	F	L	A	J	D	--	I	K	B	--	E	H
	1					11		14	12			13	-1	10	2
						0		9				3		4	
						7		6							
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Raiz = 5
Livre = 8

b) utilizando apontador e alocação dinâmica - lista de apontadores



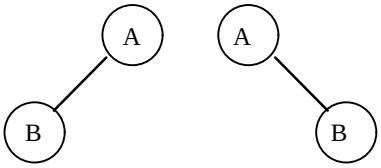
Árvores Binárias

Definição: é um conjunto finito de nós, que se apresenta vazio ou que consiste de uma **raiz** e de **duas** árvores binárias separadas, denominadas a subárvore esquerda e a subárvore direita.

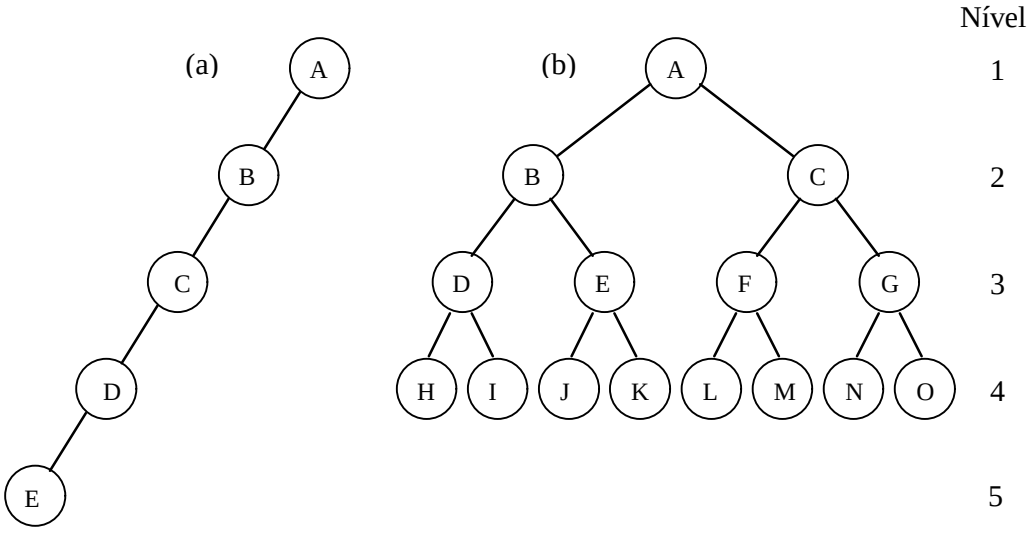
obs1.: numa árvore binária não existe nó com grau superior a dois.

obs2.: uma árvore binária completa de profundidade **k** é uma árvore binária com 2^{k-1} nós.

Não existe árvore vazia (nós com grau zero), ao passo que existe um árvore binária vazia. Se considerarmos as duas árvores abaixo como sendo árvores binárias podemos afirmar que elas são diferentes pois a primeira tem subárvore direita vazia e a segunda tem subárvore esquerda vazia. Se as considerarmos como árvores elas seriam idênticas, apenas desenhadas de maneira diferente.



Considere as árvores binárias abaixo:



Lema:

i) o número máximo de nós do nível **k** de uma árvore binária é 2^{k-1}, $k \geq 1$.

ii) o número máximo de nós numa árvore binária de profundidade **n** é 2^n-1, $n \geq 1$.

Prova lema (i): Indução em k

Base da Indução: a raiz é o único nó no nível k = 1 logo o número máximo de nós no níve k = 1 é $2^{k-1} = 2^0 = 1$.

Hipótese para Indução: provar que no nível k temos 2^{k-1} nós.

No nível k-1 temos 2^{k-2} nós. Como a árvore tem grau máximo 2, no nível k temos 2. 2^{k-2} nós, ou seja, 2^{k-1} nós.

Prova lema (ii)

O número máximo de nós numa árvore binária de profundidade **k** é de $\sum_{i=1}^{k}\left(\text{nº máximo de nós do nível i}\right)$.

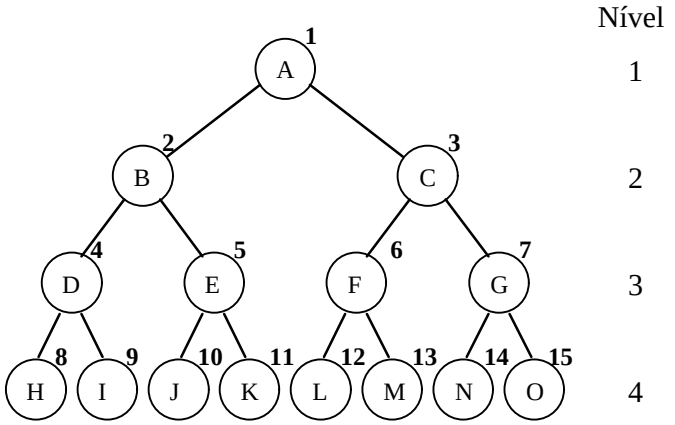
$$\sum_{i=1}^{k}\left(2^{i-1}\right) = 2^{k} - 1$$

Representação de Árvores Binárias

Estudaremos dois modos de representar árvores binárias: a representação seqüencial e a representação ligada. Na representação ligada veremos como implementar uma árvore binária utilizando um vetor de apontadores e utilizando alocação dinâmica.

a) Representação seqüencial

Nesta representação os nós são numerados por nível e da esquerda para a direita, pois dessa maneira, os nós podem ser armazenados num vetor, sendo que o nó de número k será armazenado na posição k do vetor.

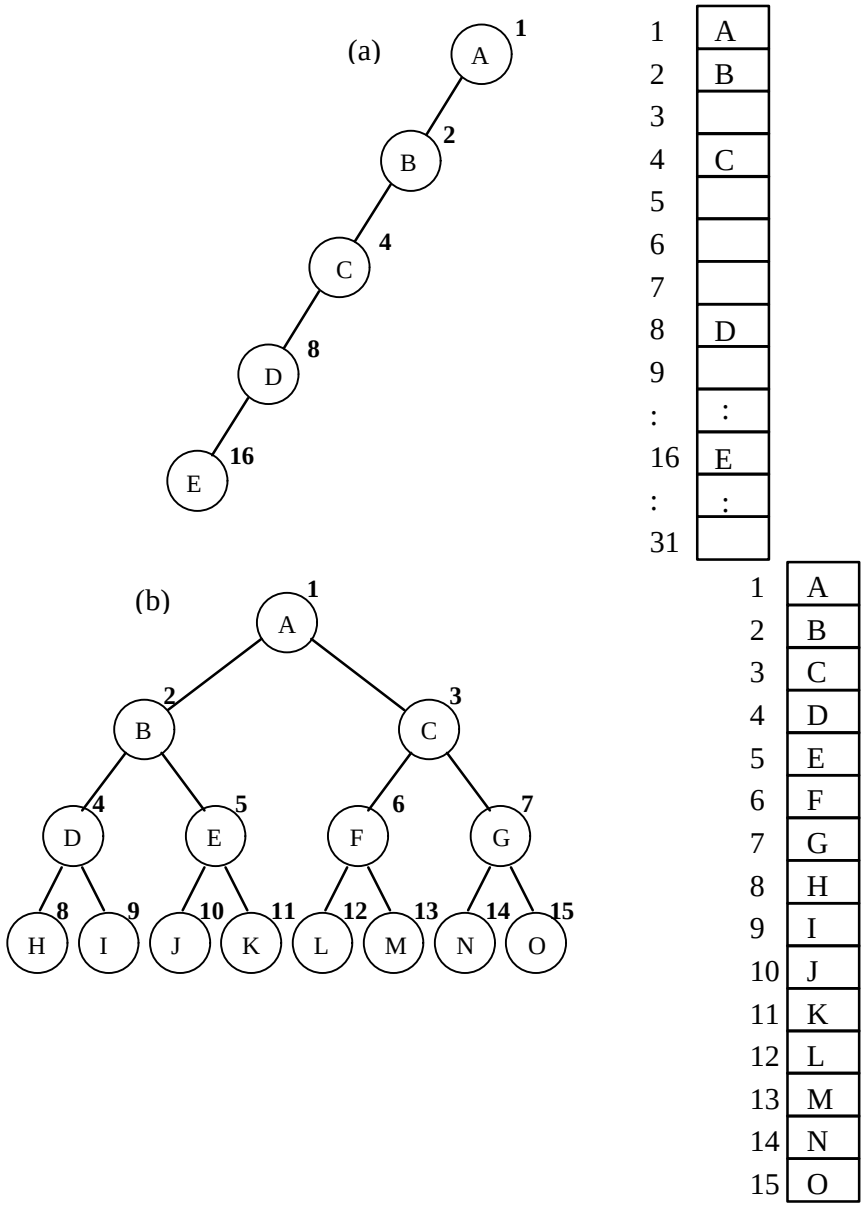


Seja uma árvore binária completa com **n** nós representados seqüencialmente.

Para qualquer nó com índice **k**, $1 \leq \mathbf{k} \leq n$ temos:

a) **Pai(k)** está em int(k/2) sendo $k \geq 1$. Se k = 0, k é a raiz
b) **FilhoEsq(k)** está em 2k, se $2k \leq n$. Se $2k > n$ então k não tem filho esquerdo.
c) **FilhoDir(k)** está em 2k + 1, se $2k + 1 \leq n$. Se $2k + 1 > n$ então k não tem filho direito.

Para árvores binárias completas, a representação é ideal pois não há perda de espaço. Para árvores assimétricas ocorrerá perda, sendo que, no pior caso, uma árvore assimétrica de profundidade **k** exigirá 2^{k-1} posições e só ocupará **k** posições (lista linear).



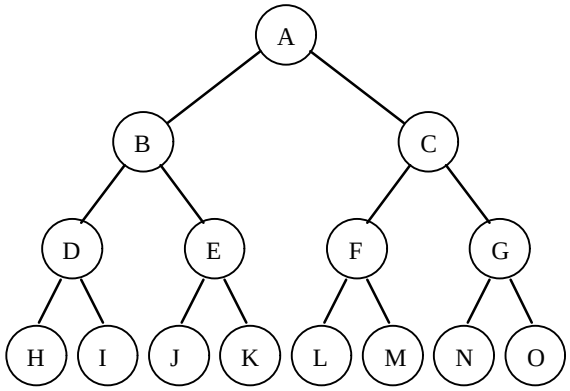
Vantagens e desvantagens da representação seqüencial

- Vantagens: estrutura e algoritmos simples, rapidez no acesso ao pai e aos filhos de um nó.
- Desvantagens: perda de espaço para árvores assimétricas (exemplo a), inserções e deleções de nós no meio da árvore exigem uma reorganização do vetor.

b) Representação Ligada

Esta representação pode ser implementada, dentre outras maneiras, através de um vetor ou através de alocação dinâmica, ambos modos utilizando apontadores (por isso chamada de representação ligada).

b.1) Utilizando um vetor

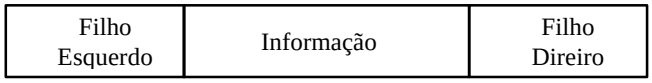


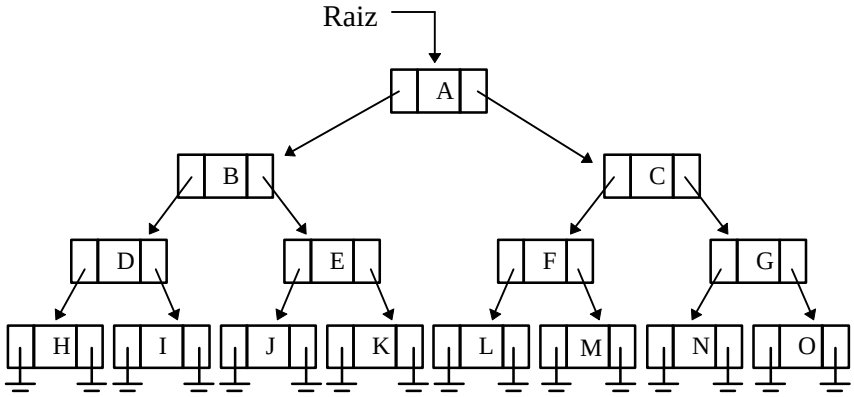
	Info	Esq	Dir
1	I		
2	B	14	7
3	J		
4	M		
5	C	9	12
6	L		
7	E	3	8
8	K		
9	F	6	4
10	A	2	5
11	O		
12	G	13	11
13	N		
14	D	15	1
15	H		

Raiz = 10

b.2) Utilizando alocação dinâmica

Esquema da estrutura de dados





As estruturas acima apresentadas dificultam a determinação do pai de um nó. Se for essencial este tipo de acesso pode-se adicionar mais um campo de modo a armazenar uma referência ao pai do nó.

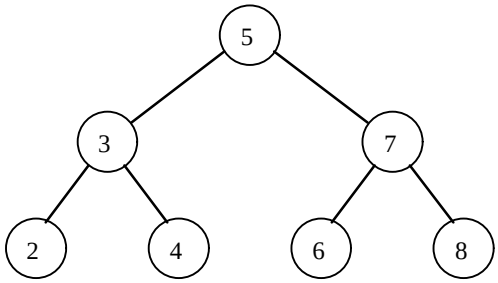
Operações em Árvores Binárias

Para construirmos uma árvore binária devemos definir as seguintes operações:

- CriaArv(x): cria uma árvore binária de um único nó com informação **x** e devolve um apontador para esta árvore.

- InsEsq(p,x): cria um novo nó, contendo a informação **x**, a esquerda do nó apontado por **p**. Se p ≠ Null ocorrerá um erro.
- InsDir(p,x): cria um novo nó, contendo a informação **x**, a direita do nó apontado por **p**. Se p ≠ Null ocorrerá um erro.
- Info(p): retorna a informação contida no nó apontado por p.
- FilhoEsq(p): retorna o apontador para o filho esquerdo de p.
- FilhoDir(p): retorna o apontador para o filho direito de p.
- Irmão(p): retorna o apontador para o irmão de p.
- Pai(p): retorna o apontador para o pai de p.

O Programa abaixo implementa as operações descritas e cria a seguinte árvore binária (arquivo Ex501.cpp):




```
#include <iostream.h>
struct No
{
  int Info;
  struct No *Dir, *Esq, *Pai;
};
struct No *Raiz;

struct No *CriaArv(int x)
{
  struct No *p;
  p = new struct No;
  p->Info = x;
  p->Esq = p->Dir = p->Pai = NULL;
  return(p);
}

void InsEsq(struct No *p, int x)
{
  if ( p == NULL)cout << "Erro: Árvore Vazia";
  else if (p->Esq != NULL ) cout << "Erro: inserção incorreta";
  else p->Esq = CriaArv(x);
}

void InsDir(struct No *p, int x)
{
  if ( p == NULL)cout << "Erro: Árvore Vazia";
  else if (p->Esq != NULL ) cout << "Erro: inserção incorreta";
  else p->Dir = CriaArv(x);
}

int Info(struct No *p)
{
```

```
  if (p != NULL ) return(p->Info);
  return(-1);
}

struct No* FilhoEsq(struct No *p)
{
  if (p != NULL ) return(p->Esq);
  return(NULL);
}

struct No* FilhoDir(struct No *p)
{
  if (p != NULL ) return(p->Dir);
  return(NULL);
}

struct No* Pai(struct No *p)
{
  if (p != NULL ) return(p->Pai);
  return(NULL);
}

struct No* Irmao(struct No *p)
{
  struct No *q;
  if (p != NULL )
  {
       q = p->Pai;
       if (q == NULL ) return NULL; //p é a raiz
       if ( q->Esq == p ) return(q->Dir);
       else return(q->Esq);
  }
  return(NULL);
}

void main()
{
  struct No *pe,*pd;

  Raiz = CriaArv(5);
  InsEsq(Raiz,3);
  InsDir(Raiz,7);
  pe = FilhoEsq(Raiz);
  pd = FilhoDir(Raiz);
  InsEsq(pe,2);
  InsDir(pe,4);
  InsEsq(pd,6);
  InsDir(pd,8);
}
```

Percursos em Árvores Binárias

Para acessarmos os elementos de uma árvore binária devemos percorrer os seus nós através das referências que temos para os seus filhos esquerdo e direito. Para que este acesso não seja desordenado e aleatório e para que possamos desenvolver os algoritmos que exploram esta estrutura devemos estabelecer algumas regras para percorrer as árvores binárias.

Tomaremos a seguinte convenção:

- **E** representa percorrer a subárvore esquerda de um nó
- **P** representa o acesso a um nó (impressão, consulta, alteração, etc.)
- **D** representa percorrer a subárvore direita de um nó

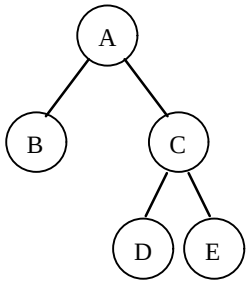
Podemos formar seis combinações possíveis: EPD, EDP, PED, PDE, DPE, DEP.

Adotando a norma de percorrer primeiro a árvore esquerda, temos três combinações:

- **EPD** - em ordem (in order)
- **EDP** - pós ordem (pos order)
- **PED** - pré ordem (pre order)

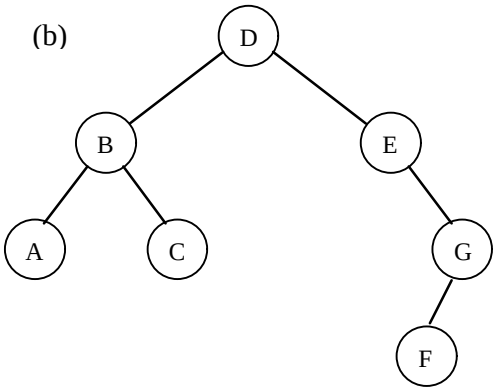
Exemplos

(a)



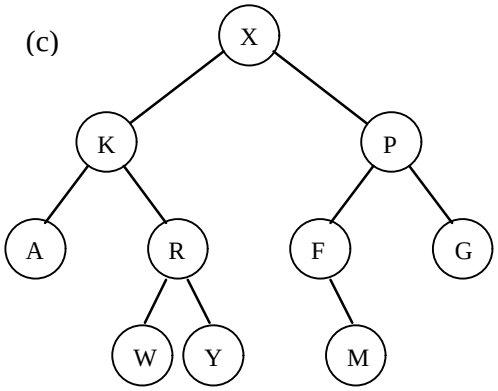

```
EPD(in  order): B A D C E

PED(pre order): A B C D E

EDP(pos order): B D E C A
```

(b)



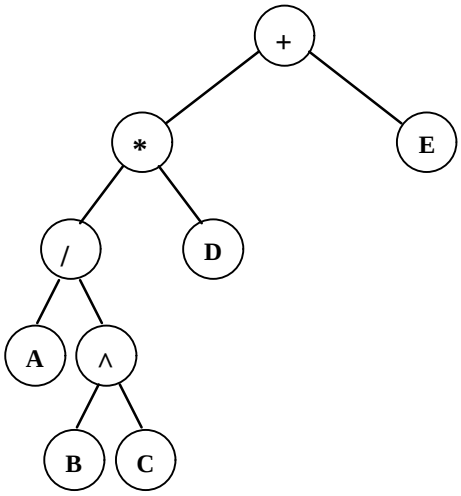

```
in order
EPD: A B C D E F G

pre order
PED: D B A C E G F

pos order
EDP: A C B F G E D
```

(c)




```
in order
EPD: A K W R Y X F M P G

pre order
PED: X K A R W Y P F M G

pos order
EDP: A W Y R K M F G P X
```

(d)




```
in order
EPD: A / B ^ C * D + E

pre order
PED: + * / A ^ B C D E

pos order
EDP: A B C ^ / D * E +
```

obs: repare que existe uma correspondência entre percorrer as árvores em ordem, pós-ordem e pré-ordem com a notação infixa, pós fixa e pré fixa.

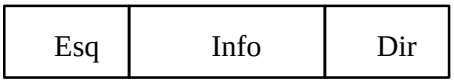
Abaixo temos os algoritmos in order, pré order e pós order para percurso em árvores binárias:

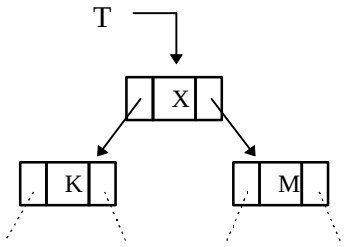
```
Algoritmo InOrder(T)
  Se T ≠ NULL então
    InOrder(T->Esq)
    Escreva(T->Info)
    InOrder(T->Dir)
  fimse

Algoritmo PreOrder(T)
  Se T ≠ NULL então
    Escreva(T->Info)
    PreOder(T->Esq)
    PreOrder(T->Dir)
  fimse

Algoritmo PosOrder(T)
  Se T ≠ NULL então
    PosOrder(T->Esq)
    PosOrder(T->Dir)
    Escreva(T->Info)
  fimse
```

A implementação destes algoritmos depende da estrutura de dados adotada. Abaixo mostraremos três formas de implementação:

1) Algoritmos para percurso em árvores binárias utilizando representação ligada com alocação dinâmica e apontadores (arquivo Ex502.cpp).

Esquema de estrutura ligada com alocação dinâmica





continua a árvore ...

```
#include <iostream.h>
#include <string.h>
struct No
{
  char Info[10];
  struct No *Dir, *Esq;
};
struct No *Raiz;

void InOrder(struct No *T)
{
  if ( T != NULL )
  {
       InOrder(T->Esq);
       cout << T->Info << " ";
       InOrder(T->Dir);
  }
}
void PreOrder(struct No *T)
{
  if ( T != NULL )
  {
       cout << T->Info << " ";
       PreOrder(T->Esq);
       PreOrder(T->Dir);
  }
}
void PosOrder(struct No *T)
{
  if ( T != NULL )
  {
       PosOrder(T->Esq);
       PosOrder(T->Dir);
       cout << T->Info << " ";
  }
}
void monta_arv_exemplo()
{
  struct No *p,*pe,*pd;
  Raiz = p = (struct No*) new struct No;
  strcpy(Raiz->Info,"F"); Raiz->Esq=NULL; Raiz->Dir=NULL;
  pe = (struct No*) new struct No;
  strcpy(pe->Info,"B"); pe->Esq=NULL; pe->Dir=NULL;
```

```
  pd = (struct No*) new struct No;
  strcpy(pd->Info,"G"); pd->Esq=NULL; pd->Dir=NULL;
  p->Esq = pe; p->Dir = pd;

  p = Raiz->Esq;  //p aponta p/ o elemento B da arvore
  pe = (struct No*) new struct No;
  strcpy(pe->Info,"A"); pe->Esq=NULL; pe->Dir=NULL;
  pd = (struct No*) new struct No;
  strcpy(pd->Info,"D"); pd->Esq=NULL; pd->Dir=NULL;
  p->Esq = pe; p->Dir = pd;

  p = pd; //p aponta p/ o elemento D da arvore
  pe = (struct No*) new struct No;
  strcpy(pe->Info,"C"); pe->Esq=NULL; pe->Dir=NULL;
  pd = (struct No*) new struct No;
  strcpy(pd->Info,"E"); pd->Esq=NULL; pd->Dir=NULL;
  p->Esq = pe; p->Dir = pd;

  p = Raiz->Dir;  //p aponta p/ o elemento G da arvore
  pd = (struct No*) new struct No;
  strcpy(pd->Info,"I"); pd->Esq=NULL; pd->Dir=NULL;
  p->Dir = pd;

  p = pd;  //p aponta p/ o elemento I da arvore
  pe = (struct No*) new struct No;
  strcpy(pe->Info,"H"); pe->Esq=NULL; pe->Dir=NULL;
  p->Esq = pe;
}
void main()
{
  Raiz = NULL;
  monta_arv_exemplo();
  InOrder(Raiz); cout << endl;
  PreOrder(Raiz); cout << endl;
  PosOrder(Raiz); cout << endl;
}
```

Os algoritmos de percurso em árvore foram implementados utilizando recursividade, entretanto podemos escrevê-los de forma não recursiva como mostrado abaixo para o caso do percurso In Order (arquivo Ex503.cpp).

```
void InOrderNR(struct No *T)
{
  struct Pilha s;
  struct No *p;
  InicPilha(s);
  p = T;
  do{
       while ( p != NULL ) {
            Empilha(s,p);
            p = p->Esq;
       }
       if ( !PilhaVazia(s) ) {
            Desempilha(s,p);
            cout << p->Info << " ";
            p = p->Dir;
       }
  } while ( !PilhaVazia(s) || p != NULL );
}
```

2) Algoritmos para percurso em árvores binárias utilizando representação ligada com vetor e apontadores (arquivo Ex504.cpp).

```
#include <iostream.h>
#include <string.h>
#define MAX_NO 12
struct No {
  char Info[10];
  int Dir,Esq;
};
struct No Arv[MAX_NO];
int Raiz, Livre;

void InOrder(int T)
{
  if ( T != -1 ) {
       InOrder(Arv[T].Esq);
       cout << Arv[T].Info << " ";
       InOrder(Arv[T].Dir);
  }
}
void PreOrder(int T)
{
  if ( T != -1 ) {
       cout << Arv[T].Info << " ";
       PreOrder(Arv[T].Esq);
       PreOrder(Arv[T].Dir);
  }
}
void PosOrder(int T)
{
  if ( T != -1 ) {
       PosOrder(Arv[T].Esq);
       PosOrder(Arv[T].Dir);
       cout << Arv[T].Info << " ";
  }
}
void monta_arv_exemplo()
{
  Raiz = 4;
  Livre = 7;
  strcpy(Arv[0].Info,"E");  Arv[0].Esq=-1; Arv[0].Dir=-1;
  strcpy(Arv[1].Info,"G");  Arv[1].Esq=-1; Arv[1].Dir=10;
  strcpy(Arv[2].Info,"H");  Arv[2].Esq=-1; Arv[2].Dir=-1;
  strcpy(Arv[3].Info,"D");  Arv[3].Esq=9;  Arv[3].Dir=0;
  strcpy(Arv[4].Info,"F");  Arv[4].Esq=8;  Arv[4].Dir= 1;
  strcpy(Arv[5].Info,"A");  Arv[5].Esq=-1; Arv[5].Dir=-1;
  strcpy(Arv[6].Info,"--"); Arv[6].Esq=11; Arv[6].Dir=-1;
  strcpy(Arv[7].Info,"--"); Arv[7].Esq=6;  Arv[7].Dir=-1;
  strcpy(Arv[8].Info,"B");  Arv[8].Esq=5;  Arv[8].Dir=3;
  strcpy(Arv[9].Info,"C");  Arv[9].Esq=-1; Arv[9].Dir=-1;
  strcpy(Arv[10].Info,"I"); Arv[10].Esq=2; Arv[10].Dir=-1;
  strcpy(Arv[11].Info,"--");Arv[11].Esq=-1;Arv[11].Dir=-1;
}
void main()
{
  Raiz = NULL;
  monta_arv_exemplo();
  InOrder(Raiz); cout << endl;
  PreOrder(Raiz); cout << endl;
  PosOrder(Raiz); cout << endl;
}
```

3) Algoritmos para percurso em árvores binárias utilizando representação seqüencial (arquivo Ex505.cpp).

Relembrando. Para uma árvore binária representada seqüencialmente temos:

a) **Pai(k)** está em int(k/2)
b) **FilhoEsq(k)** está em 2k
c) **FilhoDir(k)** está em 2k+1

Como na linguagem C os vetores iniciam na posição 0 devemos adaptar estas regras para esta particularidade da linguagem. Assim sendo temos:

a) **Pai(k)** está em int((k-1)/2)
b) **FilhoEsq(k)** está em 2k+1
c) **FilhoDir(k)** está em 2k+2

```
#include <iostream.h>
#include <string.h>
#define MAX_NO 30
struct No
{
  char Info[10];
  int ExisteNo; //indica se o nó existe
};
struct No Arv[MAX_NO];

void InOrder(int T)
{
  if ( Arv[T].ExisteNo != 0 )
  {
       InOrder(2*T + 1);
       cout << Arv[T].Info << " ";
       InOrder(2*T + 2);
  }
}
void PreOrder(int T)
{
  if ( Arv[T].ExisteNo != 0 )
  {
       cout << Arv[T].Info << " ";
       PreOrder(2*T + 1);
       PreOrder(2*T + 2);
  }
}
void PosOrder(int T)
{
  if ( Arv[T].ExisteNo != 0 )
  {
       PosOrder(2*T + 1);
       PosOrder(2*T + 2);
       cout << Arv[T].Info << " ";
  }
}
void monta_arv_exemplo()
{
  strcpy(Arv[ 0].Info,"F"); Arv[ 0].ExisteNo = 1;
  strcpy(Arv[ 1].Info,"B"); Arv[ 1].ExisteNo = 1;
  strcpy(Arv[ 2].Info,"G"); Arv[ 2].ExisteNo = 1;
  strcpy(Arv[ 3].Info,"A"); Arv[ 3].ExisteNo = 1;
  strcpy(Arv[ 4].Info,"D"); Arv[ 4].ExisteNo = 1;
  strcpy(Arv[ 5].Info,"-"); Arv[ 5].ExisteNo = 0;
  strcpy(Arv[ 6].Info,"I"); Arv[ 6].ExisteNo = 1;
  strcpy(Arv[ 7].Info,"-"); Arv[ 7].ExisteNo = 0;
```

```
  strcpy(Arv[ 8].Info,"-"); Arv[ 8].ExisteNo = 0;
  strcpy(Arv[ 9].Info,"C"); Arv[ 9].ExisteNo = 1;
  strcpy(Arv[10].Info,"E"); Arv[10].ExisteNo = 1;
  strcpy(Arv[11].Info,"-"); Arv[11].ExisteNo = 0;
  strcpy(Arv[12].Info,"-"); Arv[12].ExisteNo = 0;
  strcpy(Arv[13].Info,"H"); Arv[13].ExisteNo = 1;
  strcpy(Arv[14].Info,"-"); Arv[14].ExisteNo = 0;
  for(int k=15; k < 30; k++) Arv[k].ExisteNo = 0;
}
void main()
{
  monta_arv_exemplo();      //o índice 0 indica a raiz da árvore
  InOrder(0); cout << endl; //binária na representação seqüencial
  PreOrder(0); cout << endl;
  PosOrder(0); cout << endl;
}
```

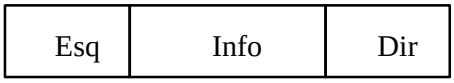
Representação de Árvores por Árvores Binárias

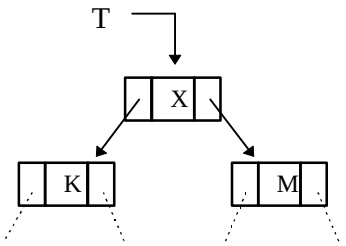
Podemos representar qualquer árvore sob a forma de árvore binária. Isto é importante pois os métodos utilizados para representação de árvores normalmente têm algumas características indesejáveis, como por exemplo nós de tamanho variável.

Ao chegar á representação de árvore binária para árvore vamos implicitamente nos servir do fato de que não tem importância a seqüência dos filhos de um nó.

Originalmente a estrutura de árvore binária foi definida do seguinte dado:

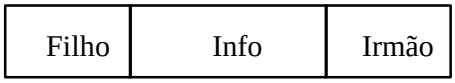
Esquema de estrutura ligada com alocação dinâmica





continua a árvore ...

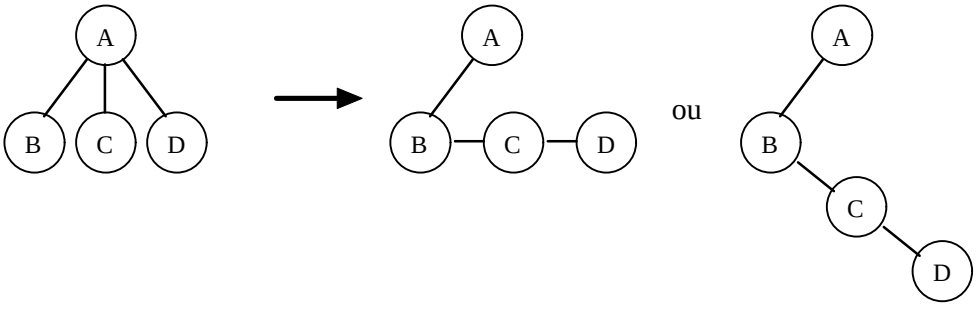
Para representar uma árvore como árvore binária, mudamos apenas a **relação** entre os nós da forma abaixo (note que a estrutura não foi alterada). Repare também que, com esta mudança, o filho direito da raiz da árvore binária estará sempre vazio pois, neste caso, a raiz não possui irmãos.

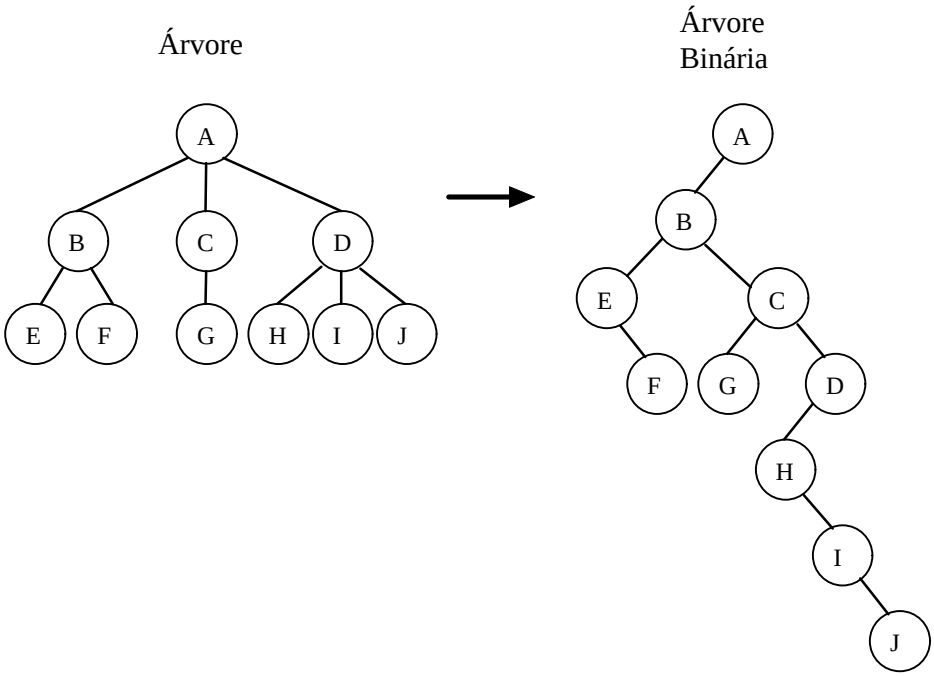


Exemplos

Árvore

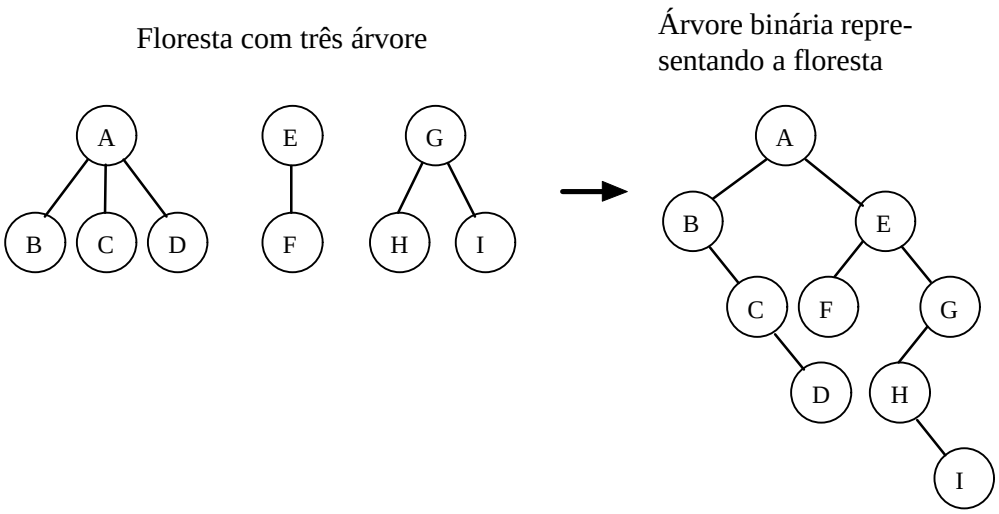
Árvore Binária





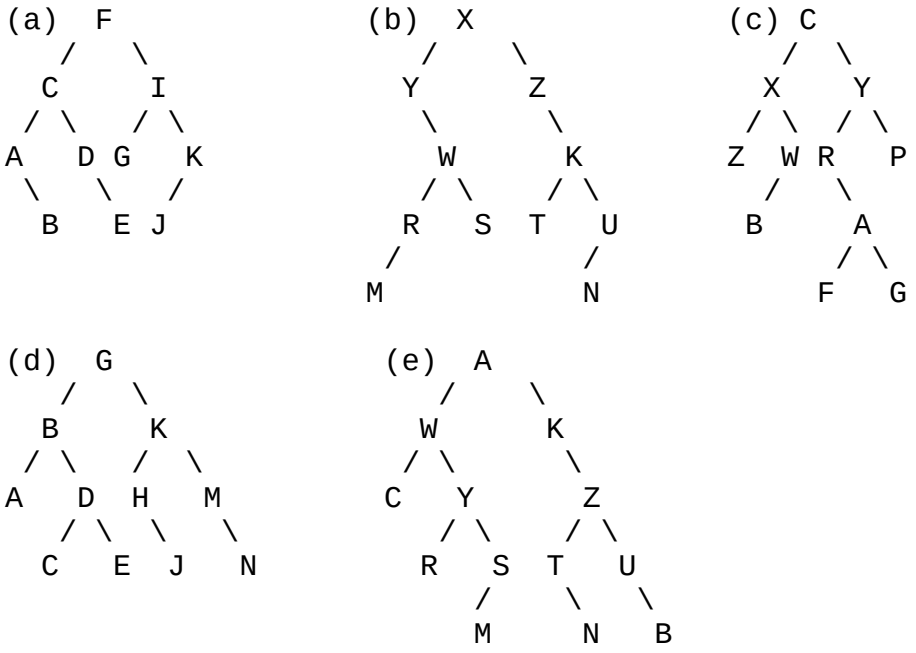
Por outro lado, uma floresta pode ser transformada numa única árvore binária utilizando para isto o campo filho direito (no caso irmão) da raiz da árvore binária.

Exemplo:



Exercícios

Resolva os exercícios 5.01, 5.02 e 5.03 utilizando as árvores binárias abaixo:



5.01) Represente as árvores binárias acima utilizando a representação seqüencial.

5.02) Represente as árvores binárias acima utilizando a representação ligada com vetor (distribua os dados da árvore da maneira que você desejar dentro do vetor).

5.03) Qual a seqüência de nós visitados se percorrermos as árvores binárias acima segundo os algoritmos in order, pre order e pos order ?

5.04) Altere o programa Ex502.cpp (árvore binária com alocação dinâmica – representação ligada) para armazenar um cadastro de pessoas contendo Nome (string 30), idade (inteiro) e sexo (caracter).

5.05) Altere o programa Ex504.cpp (árvore binária com vetor – representação ligada) para armazenar um cadastro de CDs contendo Titulo do CD (string 20), número de músicas (inteiro) e preço (real).

5.06) Altere o programa Ex505.cpp (árvore binária – representação seqüencial) para armazenar um cadastro de Livros contendo Titulo do livro (string 20), número de páginas (inteiro) e preço (real).

5.07) Elaborar função para contar o número de nós de uma árvore binária.

5.08) Elaborar função para contar o número de folhas de uma árvore binária. Utilize a representação ligada com alocação dinâmica.

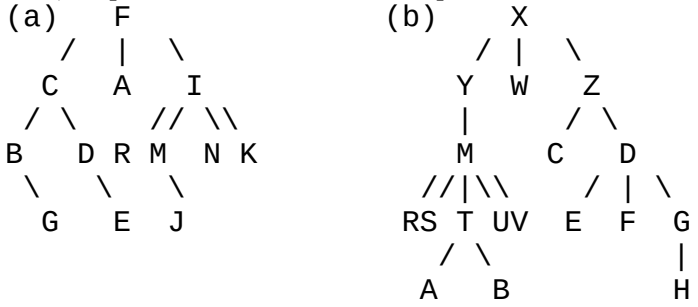
5.09) Elaborar uma função para determinar a altura de uma árvore binária.

5.10) Elaborar uma função para imprimir apenas os nós de um determinado nível de uma árvore binária.

5.11) Elaborar uma função para encontrar o pai de um nó (considerar que esta estrutura não possui um apontador para o nó pai).

5.12) Elaborar uma função para encontrar o irmão de um nó (considerar que esta estrutura não possui um apontador para o nó pai).

5.13) Represente as árvores abaixo por meio de árvores binárias



Respostas

5.01) representação seqüencial

árvore (a)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F	C	I	A	D	G	K	-	B	-	E	-	-	J	-

5.02) representação ligada com vetor

árvore (a)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Info	D	--	C	J	F	A	--	I	B	--	K	E	--	G	--
Esq	-1	9	5	-1	2	-1	1	13	-1	12	3	-1	14	-1	-1
Dir	11		0	-1	7	8		10	-1		-1	-1		-1	

Raiz = 4 Livre = 6

5.03) percorrer as árvores binárias segundo os algortimos In, Pre e Pos Order

árvore (a)

```
In  Order(EPD): A B C D E F G I J K
Pre Order(PED): F C A B D E I G K J
Pos Order(EDP): B A E D C G J K I F
```

5.04) Este é por sua conta. Divirta-se !

Apenas uma dica. A estrutura para este exercício pode ser como a mostrada abaixo:

```
struct No {
  char Nome[30];
  int Idade;
  char Sexo;
  struct No *Dir, *Esq;
};
struct No *Raiz;
```

5.05) Este é por sua conta.

5.06) Este é por sua conta.

5.07) contar o número de nós de uma árvore binária

```
int ContaNos(struct No *T)
{
  if ( T == NULL ) return(0);
```

```
  else return( 1 + ContaNos(T->Esq) + ContaNos(T->Dir));
}
```

5.08) contar o número de nós de uma árvore binária

```
int ContaFolhas(struct No *T)
{
  if ( T == NULL ) return(0);
  else
  {
    if (T->Esq == NULL && T->Dir == NULL ) return(1);
    else return( ContaFolhas(T->Esq) + ContaFolhas(T->Dir));
  }
}
```

5.09) Este é por sua conta. Divirta-se !

5.10) Este também.

5.11) Este também.

5.12) Este também.

5.13) Este também.