3. Listas Não Lineares

Quando existe mais de um caminho possíveis pela estrutura, esta é dita não linear. Exemplos clássicos de estruturas deste tipo são as árvores e grafos (estes estudados em Algoritmos e Estruturas de Dados II).

3.1. Árvores

Uma árvore é um conjunto de nós composto de um nó especial (chamado raiz) e conjuntos disjuntos de nós subordinados ao nó raiz que são eles próprios (sub)árvores.

Terminologia

Grau de um nó: a quantidade de subárvores do nó;
Grau de uma árvore: grau máximo dentre todos os nós da estrutura;
Folhas de uma árvore: nós de grau zero;
Filhos de x: raízes das subárvores de *x*; *x* é o nó pai de seus filhos;
Ancestrais de x: todos nós no caminho desde a raiz até *x*.
Nível de x: a raiz é nível 1; se um nó está no nível *n*, seus filhos estão no nível *n*+1;
Altura de um nó folha é sempre 0 (zero);
Altura de um nó não folha: a altura máxima dentre todas suas subárvores + 1;
Altura de uma árvore é a altura de sua raiz.

Uma árvore de grau m é dita m-ária. Árvores m-árias são de difícil representação e manipulação (por exemplo, a definição de muitos ponteiros em cada nó representa um grande desperdício de espaço ocupado por ponteiros NULL). Por este motivo, árvores m-árias são geralmente representadas por árvores binárias (veja definição a seguir) sem perda de propriedades.

Em computação, árvores (e especialmente árvores binárias) são usadas para armazenar dados (chaves e outros campos de informação) em seus nós da mesma forma que listas sequenciais e listas ligadas.

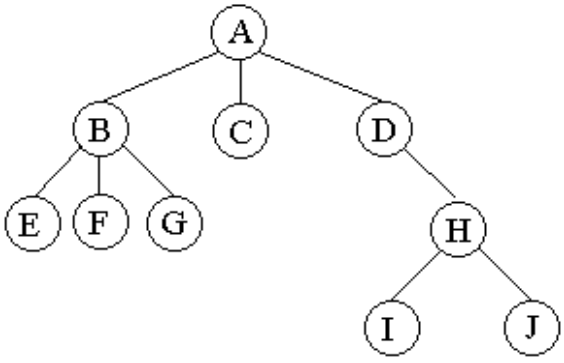
3.1.1. Árvores Binárias

Uma árvore binária é uma estrutura vazia ou um nó raiz e duas subárvores chamadas esquerda e direita, as quais são também árvores binárias (vazias ou não). É importante observar que uma árvore binária não é apenas uma árvore de grau máximo dois, pois há também a questão de ordem (esquerda e direita) de subárvores, conceito este que não existe na definição de árvore comum discutida no item anterior.

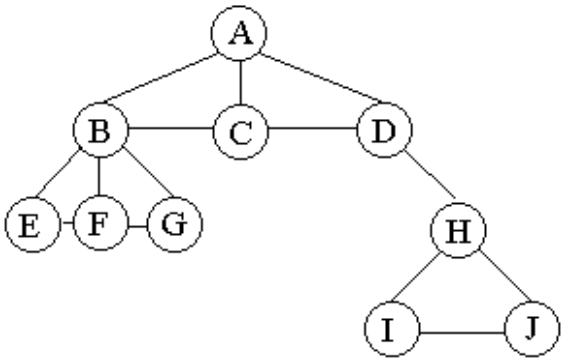
Uma vez que árvores m-árias não definem uma ordem específica entre as subárvores de cada nó, a conversão de árvore m-ária para binária é trivial: basta eleger um filho qualquer como raiz da subárvore esquerda, e os demais como subárvore direita e seus descendentes. O procedimento de conversão compreende dois passos:

(a) todos os nós irmãos da árvore m-ária são interligados;
(b) todas as conexões pai-filho são removidas, exceto a primeira de cada grupo.

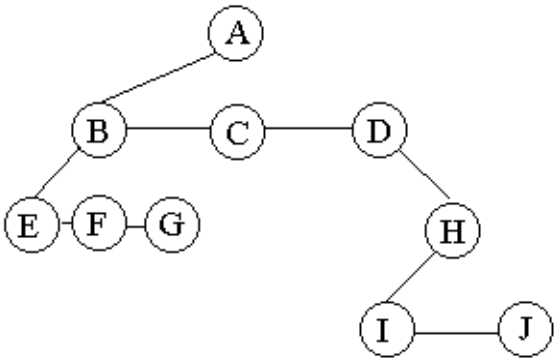
A árvore assim resultante pode ser redesenhada na posição correta (preservando-se as ligações esquerda e direita estabelecidas) formando uma árvore binária.



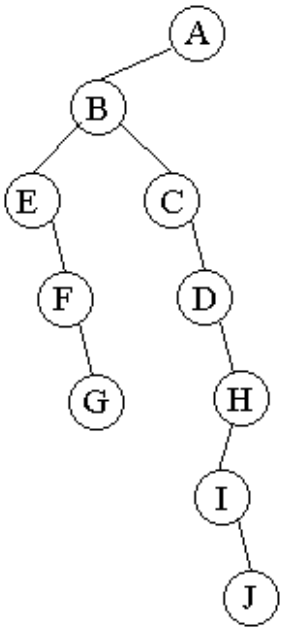
Árvore m-ária a ser convertida



Passo (a) nós irmãos são interligados



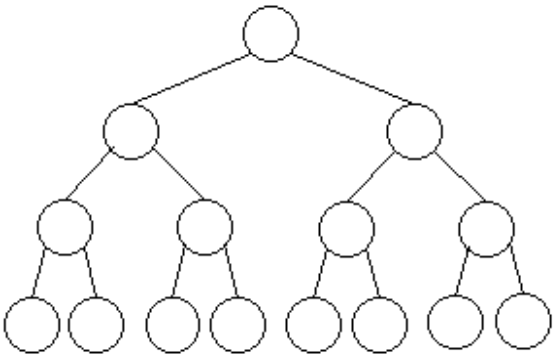
Passo (b): ligações pai-filho são removidas, exceto a primeira de cada grupo



Árvore binária resultante (as ligações pai-filho originais são mantidas)

Propriedades

- O número máximo de nós possíveis no nível *i* é 2^{i-1}



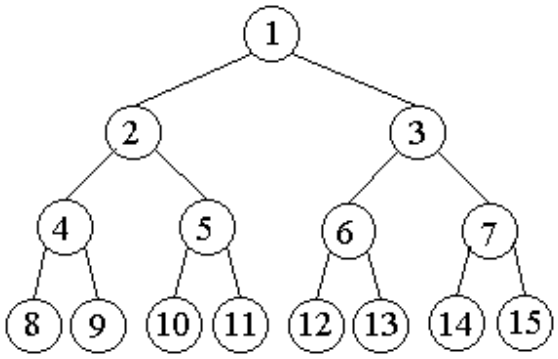
nível $1 = 2^{1-1} = 1$ nó

nível $2 = 2^{2-1} = 2$ nós

nível $3 = 2^{3-1} = 4$ nós

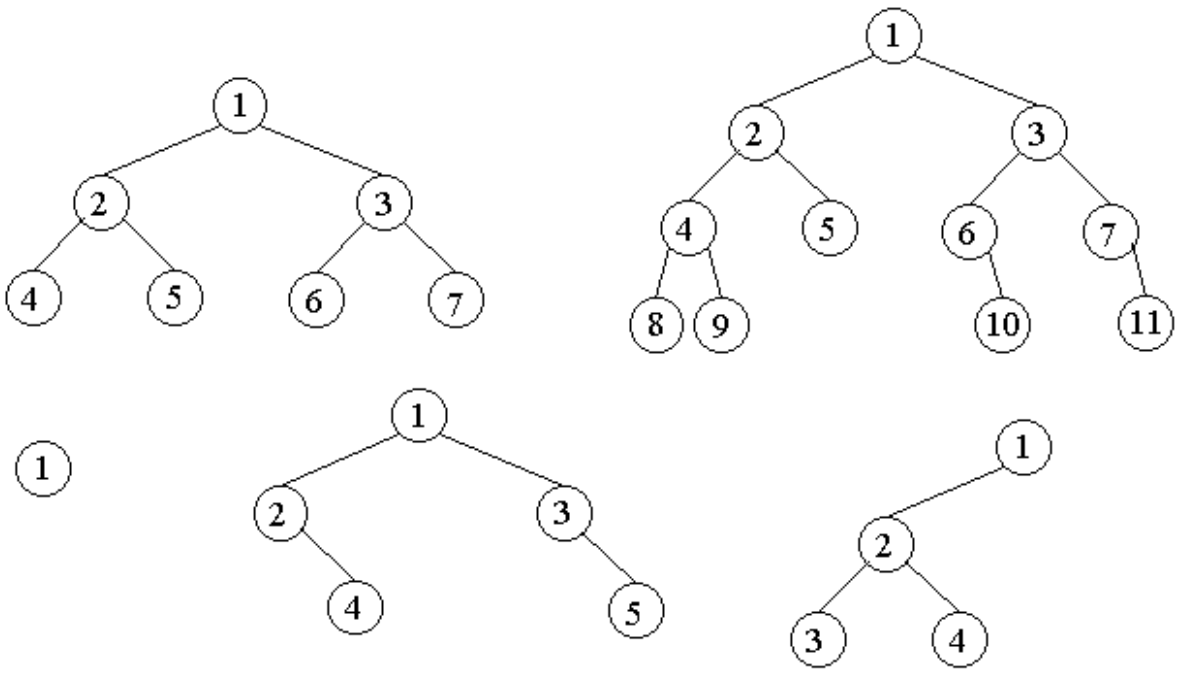
nível $4 = 2^{4-1} = 8$ nós

- Uma árvore binária de altura *h* tem no máximo 2^{h+1} -1 nós.



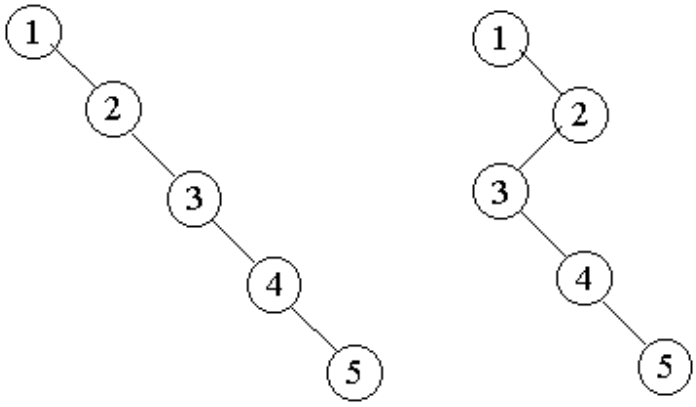
para $h = 3$, $n_{máx} = 2^4 - 1 = 15$

- A altura mínima de uma árvore binária com $n > 0$ nós é $1 + \text{chão}(\log_2 n)$
- Uma árvore binária de altura mínima é dita ***completa***.



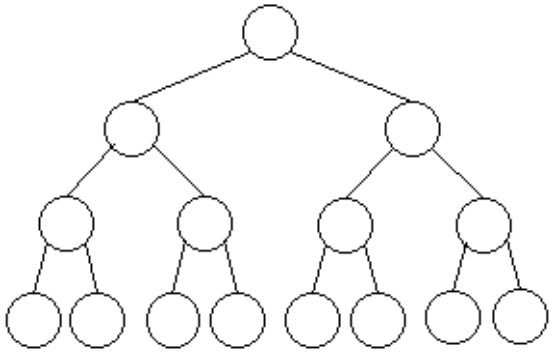
Exemplos de árvores completas. Seus nós não podem ser redistribuídos formando uma árvore de altura menor do que estas.

- Uma árvore binária de altura máxima para uma quantidade de n nós é dita ***assimétrica***. Neste caso, a altura é h=n-1 e seus nós interiores possuem exatamente uma subárvore vazia cada.



Exemplos de árvores assimétricas

- Uma árvore binária de altura *h* é ***cheia*** se possui exatamente 2^{h+1} -1 nós.



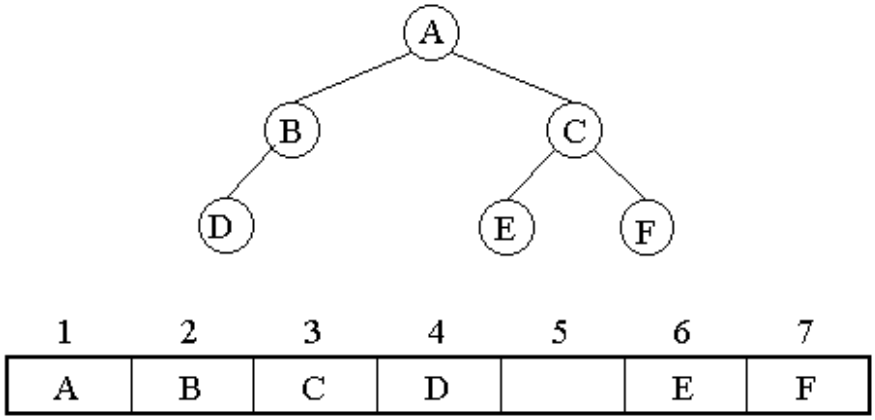
Exemplo de árvore cheia – possuindo o número máximo de nós para a sua altura.

Implementação Estática

Embora a implementação dinâmica seja mais comum, árvores binárias podem ser representadas em um vetor. Isso é especialmente útil em aplicações que não sofrem grande volume de inserções e exclusões; nos demais casos a representação dinâmica continua sendo a preferida.

Na representação estática, o vetor deve ser grande o suficiente para conter o número máximo possível de nós para a altura máxima h_{max} estabelecida, ou seja, deve ter no mínimo $2^{hmax+1} - 1$ posições. Isso é necessário porque mesmo os nós não existentes terão seu espaço reservado no vetor, já que a posição relativa de qualquer pai ou filho no vetor é fixa, definida em função das posições de seus antecessores.

Os nós da árvore são dispostos ao longo do vetor em níveis, da esquerda para a direita, começando pela raiz (que ocupa a primeira posição). Como os nós inexistentes deixam posições vazias no vetor, algum tipo de controle deve ser feito para diferenciar posições livres e ocupadas (e.g., com uso de um campo booleano).



Uma vez que a raiz ocupa a primeira posição do vetor, os outros nós têm suas posições definidas como segue:

- pai(i) = chão (i / 2) se $i == 0$, não há pai
- filho_esq(i) = 2 * i se $i > n$, não há filho esquerdo
- filho_dir(i) = 2 * i + 1 se $i > n$, não há filho direito

Implementação Dinâmica

```
typedef struct estrutura
{
        TIPOCHAVE chave;
        estrutura *esq;
        estrutura *dir;
} NO;

// Inicialização da árvore vazia
void inicializarArvore(NO* *raiz)
{
     *raiz = NULL;
}
```

```
// Verificar se árvore é vazia
bool arvoreVazia(NO* raiz)
{
        if(!raiz) return(true);
        else return(false);
}

// Inserção de um nó em árvore comum (sem ordem)

typedef enum {esq, dir} LADO;

bool inserirNo(NO* *raiz, NO* pai, TIPOCHAVE ch, LADO pos)
{
        NO* novo;
        if(pai)
        {
                if(     ((pos==esq) && (pai->esq!=NULL))||
                        ((pos==dir) && (pai->dir!=NULL)))
                {
                      return(false);
                }
        }
        novo = (NO *) malloc(sizeof(NO));
        novo->chave = ch;
        novo->esq = NULL;
        novo->dir = NULL;
        if(!pai) *raiz = novo;
        else
        {
          if(pos==esq)
                pai->esq = novo;
          else
                pai->dir = novo;
        }
}
```

Percursos em árvore binária

Convenção: as operações possíveis (determinadas pelos ponteiros existentes) são três: visitar a raiz, deslocar-se para a esquerda e deslocar-se para a direita. A esquerda sempre tem prioridade sobre direita.

Os percursos possíveis de acordo com esta convenção são:
- *Pré-ordem*: visita a raiz, esquerda e direita.
- *Em ordem*: esquerda, visita a raiz e direita.
- *Pós-ordem*: esquerda, direita e visita raiz.