Árvores

- Peof. Me. Sérgio Carlos Portari Júnior

Árvores

- São estruturas de dados adequadas para apresentação de hierarquias.

- Uma árvore é composta por um conjunto de nós.

- Existe um nó **r**, denominado *raiz*, que contém zero (nenhuma) ou mais sub-árvores, cujas raízes são ligadas diretamente a **r**.

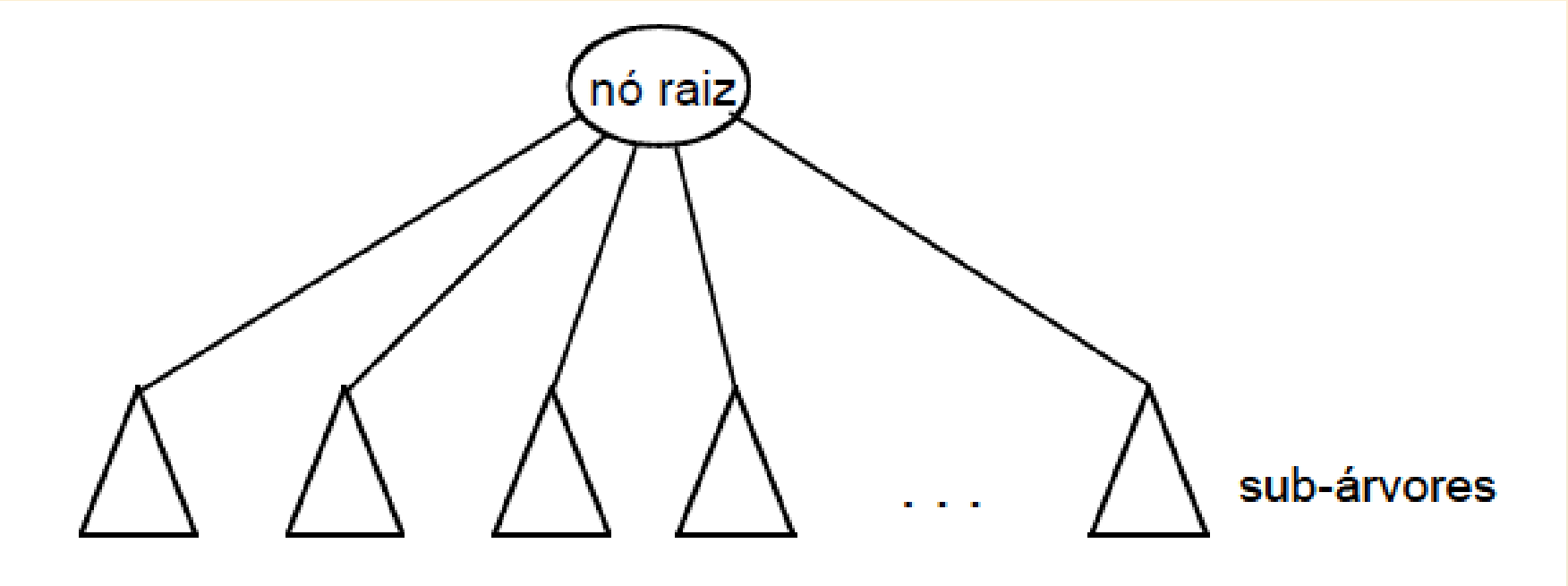
Árvores

- Esses nós raízes das sub-árvores são ditos *filhos* do nó *pai*, **r**.

- Nós com filhos são comumente chamados de *nós internos* e nós que não têm filhos são chamados de *folhas*, ou nós externos.

- É tradicional desenhar as árvores com a raiz para cima e folhas para baixo, ao contrário do que seria de se esperar.

Árvores



- Observamos que, por adotarmos essa forma de representação gráfica, não representamos explicitamente a direção dos ponteiros, subentendendo que eles apontam sempre do pai para os filhos.

Árvores

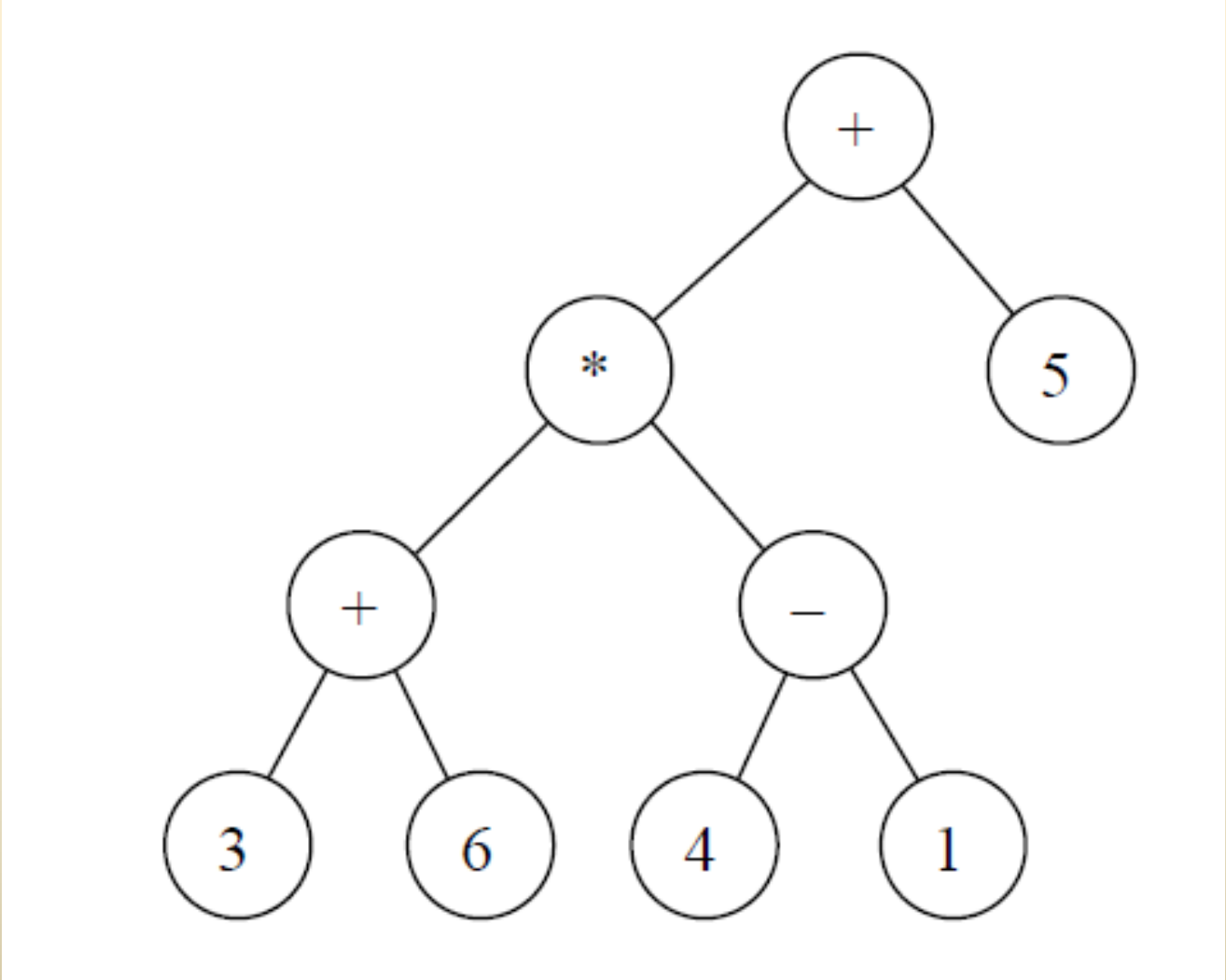
- O número de filhos permitido por nó e as informações armazenadas em cada nó diferenciam os diversos tipos de árvores existentes.

- Primeiramente, estudaremos as árvores binárias, onde cada nó tem no máximo dois filhos.

Árvore Binária

- Um exemplo de utilização de árvores binárias está na avaliação de expressões.
- Como trabalhamos com operadores que esperam um ou dois operandos, os nós da árvore para representar uma expressão têm no máximo dois filhos.
- Nessa árvore, os nós folhas representam operandos e os nós internos operadores.
- Uma árvore que representa, por exemplo, a expressão (3+6)*(4-1)+5 é ilustrada a seguir:

Árvore Binária



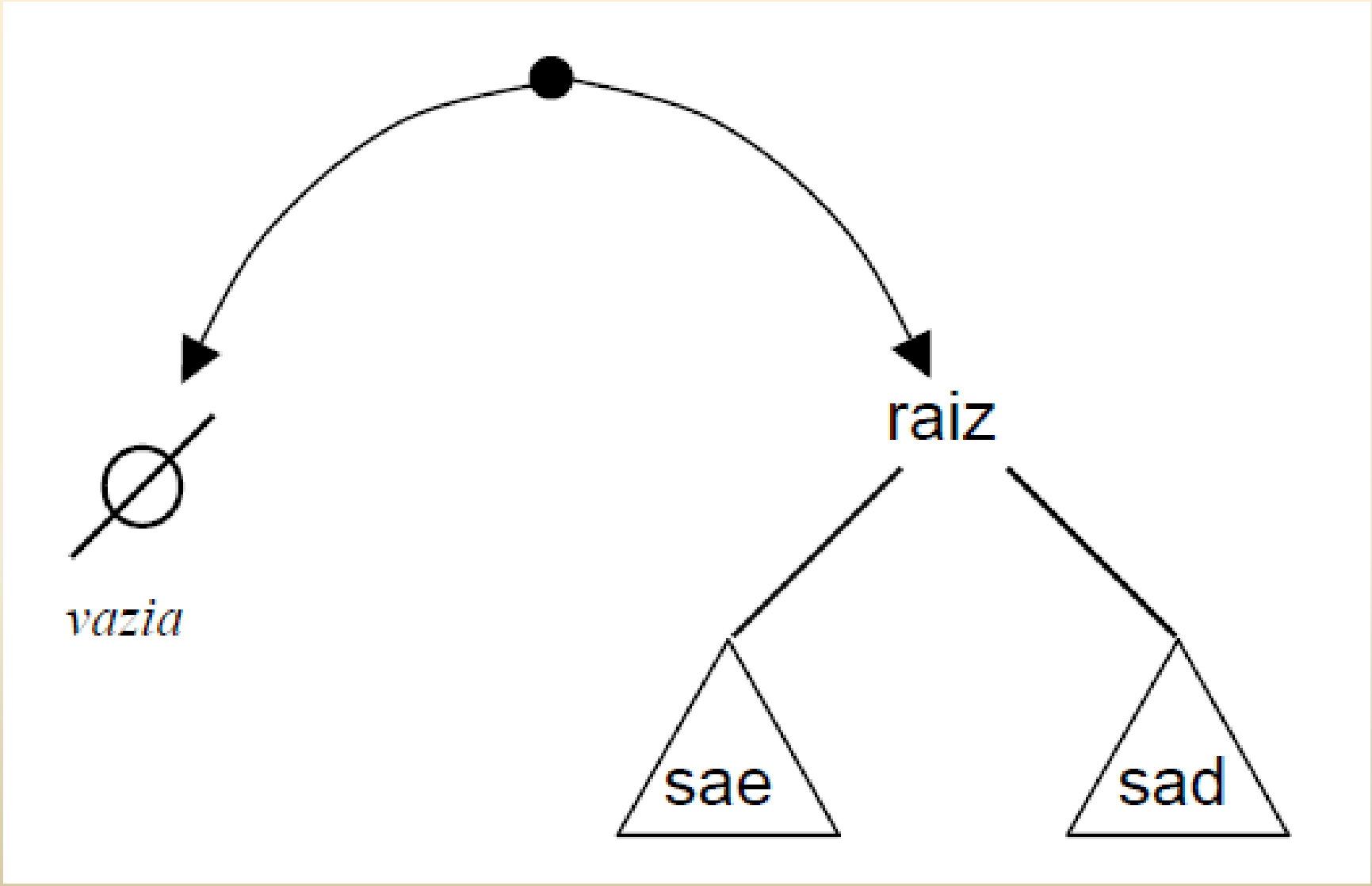
Árvore Binária

- Numa árvore binária, cada nó tem zero (nenhum), um ou dois filhos. Podemos definir uma árvore binária como sendo:

  • uma árvore vazia; ou

  • um nó raiz tendo duas sub-árvores, identificadas como a sub-árvore da direita (***sad***) e a sub-árvore da esquerda (***sae***).

Árvore Binária

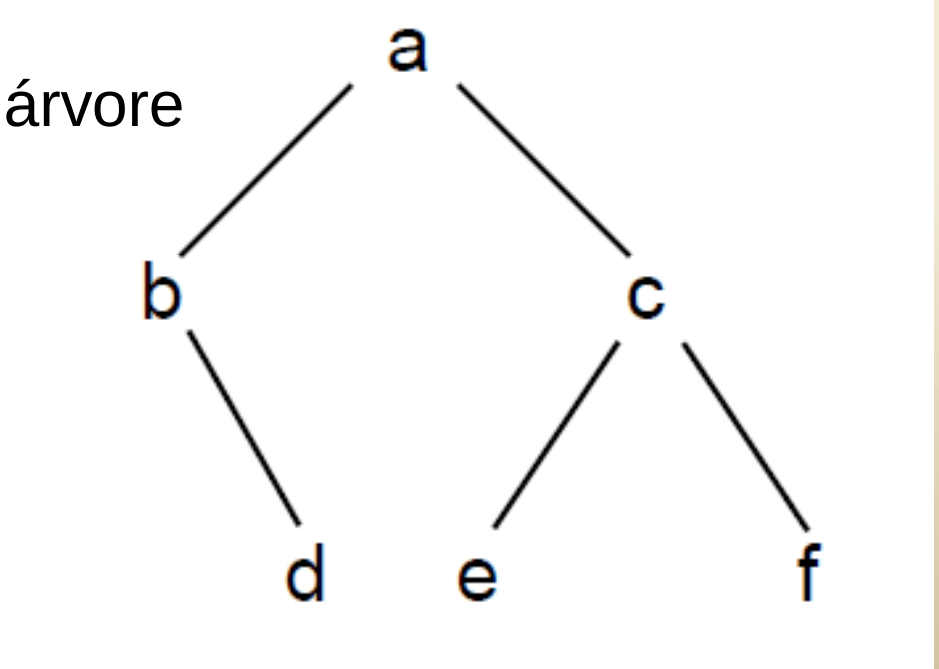


Árvore Binária

- A seguir ilustra-se uma estrutura de árvore binária. Os nós *a*, *b*, *c*, *d*, *e*, *f* formam uma árvore binária da seguinte maneira:
  - a árvore é composta do nó *a*
  - da subárvore à esquerda formada por *b* e *d*; e
  - da sub-árvore à direita formada por *c*, *e* e *f*.

- O nó *a* representa a raiz da árvore e os nós *b* e *c* as raízes das sub-árvores.

- Finalmente, os nós *d*, *e* e *f* são folhas da árvore.



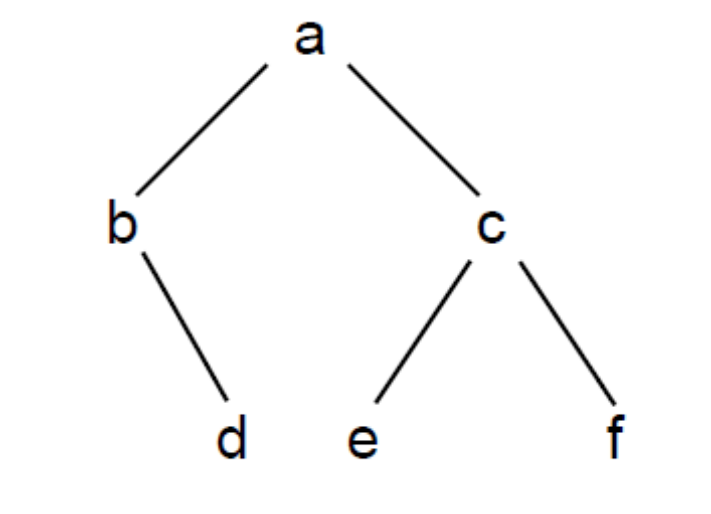
Árvore Binária

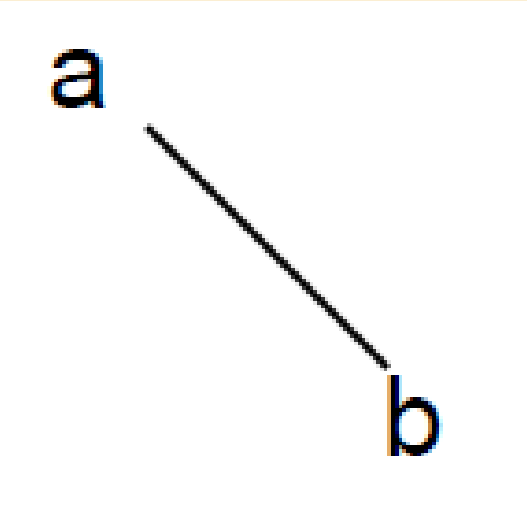
- Pela definição, uma sub-árvore de uma árvore binária é sempre especificada como sendo a *sae* ou a *sad* de uma árvore maior, e qualquer das duas sub-árvores pode ser vazia.
- Assim, as duas árvores abaixo são distintas.

Árvore Binária

- Uma propriedade fundamental de todas as árvores é que só existe **um** caminho da raiz para qualquer nó.
- Com isto, podemos definir a *altura* de uma árvore como sendo o comprimento do **caminho mais longo da raiz até uma das folhas.**
- Assim, a altura de uma árvore com um único nó raiz é zero e, por conseguinte, dizemos que a altura de uma árvore vazia é negativa e vale -1.

Árvore Binária





- Altura: 2

Altura: 1

Árvore Binária em C

- Para implementar uma árvore binária em C precisaremos criar um TAD árvore com a seguinte estrutura:

```
typedef struct arvore {
    char info;
    arvore *esq;
    arvore *dir;
} Arvore;
```

Árvore Binária em C

- Da mesma forma que uma lista encadeada, uma pilha ou uma fila; é representada por um ponteiro para o primeiro nó, a estrutura da árvore como um todo é representada por um ponteiro para o nó raiz.

- Como acontece com qualquer TAD, as operações que fazem sentido para uma árvore binária dependem essencialmente da forma de utilização que se pretende fazer da árvore.

Árvore Binária em C

- Uma operação que provavelmente deverá ser incluída em todos os casos é a inicialização de uma árvore vazia.
- Como uma árvore é representada pelo endereço do nó raiz, uma árvore vazia tem que ser representada pelo valor NULL.
- Assim, a função que inicializa uma árvore vazia pode ser simplesmente:

```
Arvore *inicializaArvore()
{
   return NULL;
}
```

Árvore Binária em C

- Para criar árvores não vazias, podemos ter uma operação que cria um nó raiz dadas a informação e suas duas sub-árvores, à esquerda e à direita.

- Essa função tem como valor de retorno o endereço do nó raiz criado e pode ser dada por:

Árvore Binária em C

```
Arvore *criaNo(char c, Arvore *sae, Arvore *sad)
{
   Arvore *a=(Arvore*)malloc(sizeof(Arvore));
   a->info = c;
   a->esq = sae;
   a->dir = sad;
   return a;
}
```

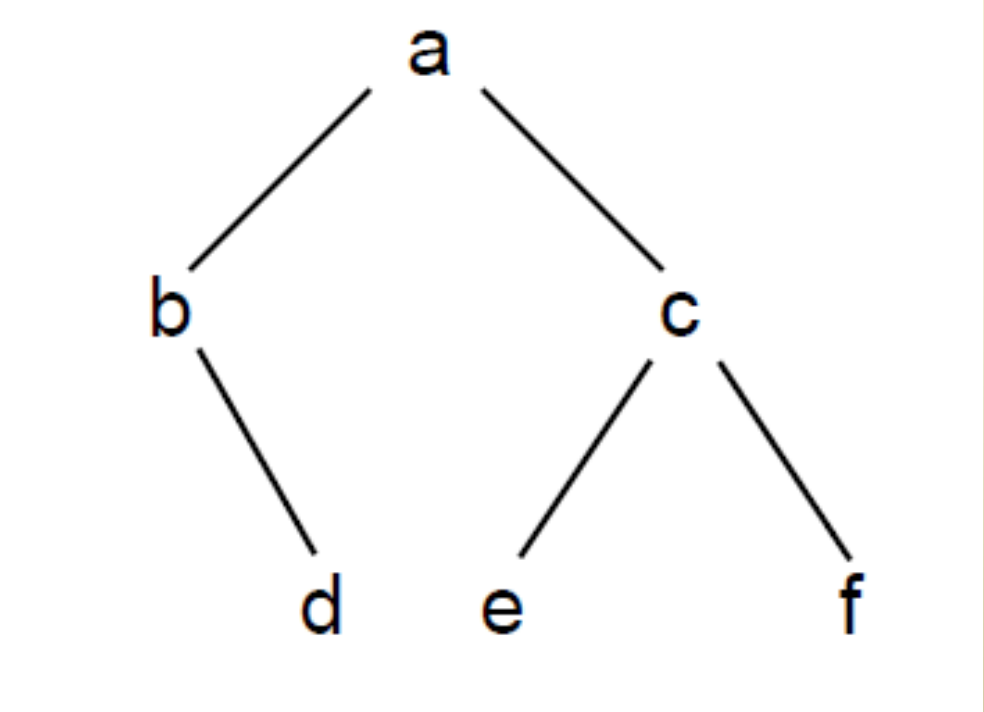
Árvore Binária em C

- As duas funções inicializaArvore e criaNo representam os dois casos da definição de árvore binária:
  - uma árvore binária (Arvore *a;) é vazia (a = inicializa();) ou
  - é composta por uma raiz e duas sub-árvores (a = criaNo(c,sae,sad);).

- Assim, com posse dessas duas funções, podemos criar árvores mais complexas.

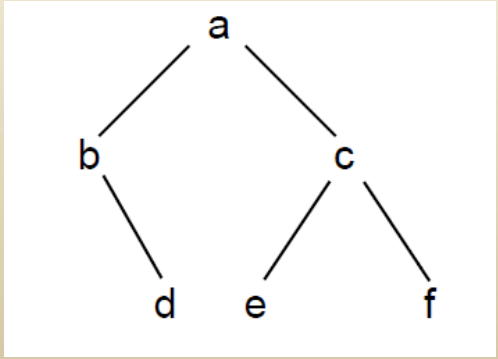
Árvore Binária em C

- Por exemplo, usando as operações inicializaArvore e criaNo, crie uma estrutura que represente a árvore da figura:



Árvore Binária em C

```
// sub-árvore com 'd'
Arvore *a1= criaNo('d',inicializaArvore(),inicializaArvore());
// sub-árvore com 'b'
Arvore *a2= criaNo('b',inicializaArvore(),a1);
// sub-árvore com 'e'
Arvore *a3= criaNo('e',inicializaArvore(),inicializaArvore());
// sub-árvore com 'f'
Arvore *a4= criaNo('f',inicializaArvore(),inicializaArvore());
// sub-árvore com 'c'
Arvore *a5= criaNo('c',a3,a4);
// árvore com raiz 'a'
Arvore *a = criaNo('a',a2,a5 );
```



Árvore Binária em C

- A mesma estrutura poderia, resumidamente, com:

```
Arvore *a = criaNo('a',
              criaNo('b',
                inicializaArvore(),
                criaNo('d', inicializaArvore(), inicializaArvore())
                ),
              criaNo('c',
                  criaNo('e', inicializaArvore(),
inicializaArvore()),
                  criaNo('f', inicializaArvore(), inicializaArvore())
                  )
            );
```

Árvore Binária em C

- Para tratar a árvore vazia de forma diferente das outras, é importante ter uma operação que diz se uma árvore é ou não vazia. Podemos ter:

```
int vazia(Arvore *a)
{
   return a==NULL;
}
```

- O retorno da função é: 0 (zero) se árvore não é vazia, e diferente de 0 (zero) se vazia.

Árvore Binária em C

- Uma outra função muito útil consiste em exibir o conteúdo da árvore. Essa função deve percorrer recursivamente a árvore, visitando todos os nós e imprimindo sua informação.
- Vimos que uma árvore binária ou é vazia ou é composta pela raiz e por duas sub-árvores.
- Portanto, para imprimir a informação de todos os nós da árvore, devemos primeiro testar se a árvore é vazia.
- Se não for, imprimimos a informação associada a raiz e chamamos (recursivamente) a função para imprimir os nós das sub-árvores.

Árvore Binária em C

```
void imprime (Arvore *a)
{
  if (!vazia(a))
  {
     printf("\n%c ", a->info); /* mostra raiz */
     imprime(a->esq); /* mostra sae */
     imprime(a->dir); /* mostra sad */
  }
}
```

Árvore Binária em C

- A função libera também é importante para liberar a memória após a utilização do programa.

```
Arvore *libera (Arvore *a)
{
  if (!vazia(a))
  {
    libera(a->esq); // libera sae
    libera(a->dir); // libera sad
    free(a); // libera raiz
  }
return NULL;
}
```

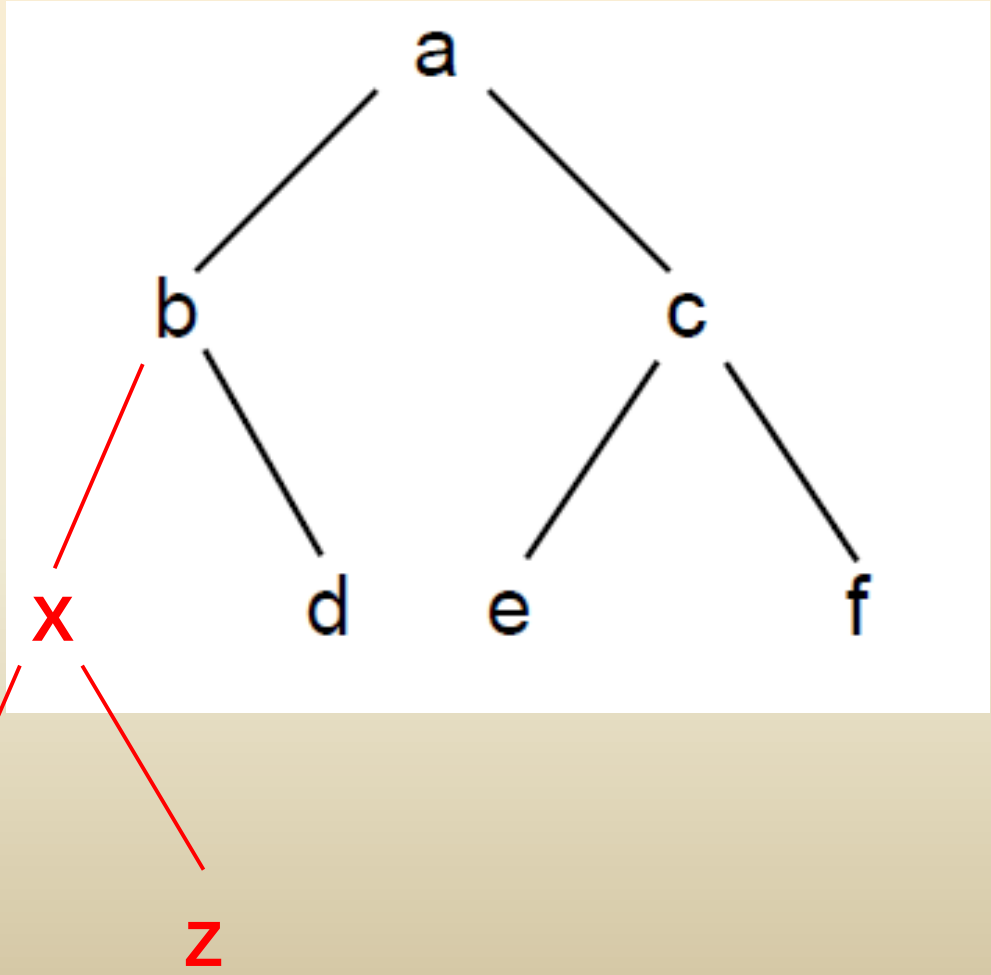
Árvore Binária em C

- Devemos notar que a definição de árvore, por ser recursiva, não faz distinção entre árvores e sub-árvores.

- Assim, **criaNo* pode ser usada para acrescentar (“enxertar”) uma sub-árvore em um ramo de uma árvore, e libera pode ser usada para remover (“podar”) uma sub-árvore qualquer de uma árvore dada.

Árvore Binária em C

- Por exemplo: para criar um novo nó na árvore:



Árvore Binária em C

```
//acrescentar nós X Y e Z à esquerda de B
    a->esq->esq = criaNo('x',
   criaNo('y',inicializaArvore(),inicializaArvore()),
   criaNo('z',inicializaArvore(),inicializaArvore())
      );
```

Árvore Binária em C

- E podemos liberar alguns outros, com:
- a->dir->esq = libera(a->dir->esq);