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);
```

```c
int isright(ARV_BIN_ENC t)
{
    if (t->father)
        return (!isleft(t));
    return (0);
}
```

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.

UNIVASF

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.

UNIVASF

Árvores Binárias – Aplicações

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

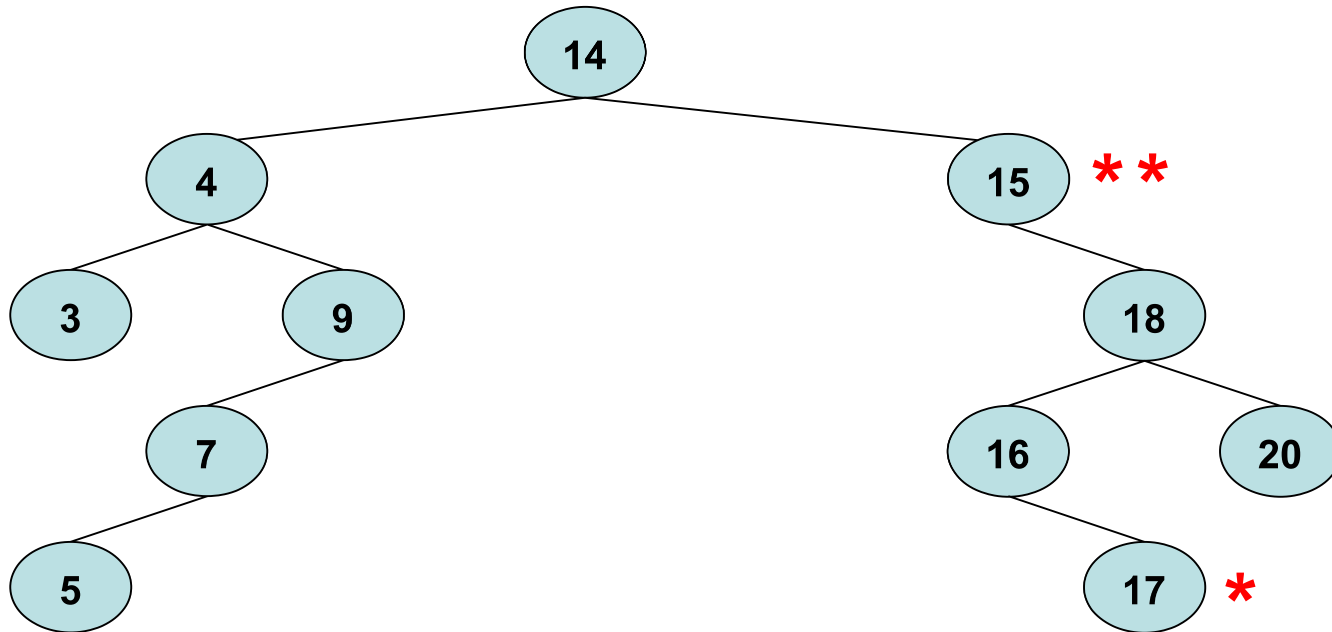
Se a subárvore não estiver vazia, compararemos o número ao conteúdo da raiz da subárvore e o processo inteiro será repetido com a subárvore.

UNIVASF

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.

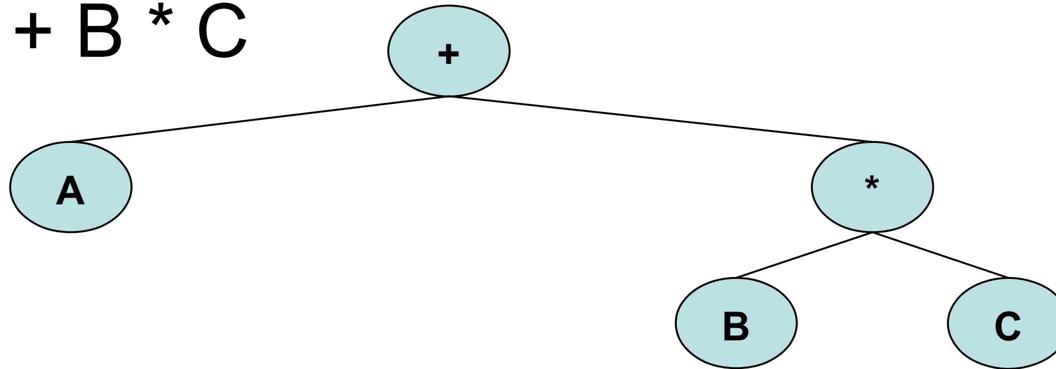
Um nó representando um operador é um nó que não é folha, enquanto um nó representando um operando é um folha.

UNIVASF

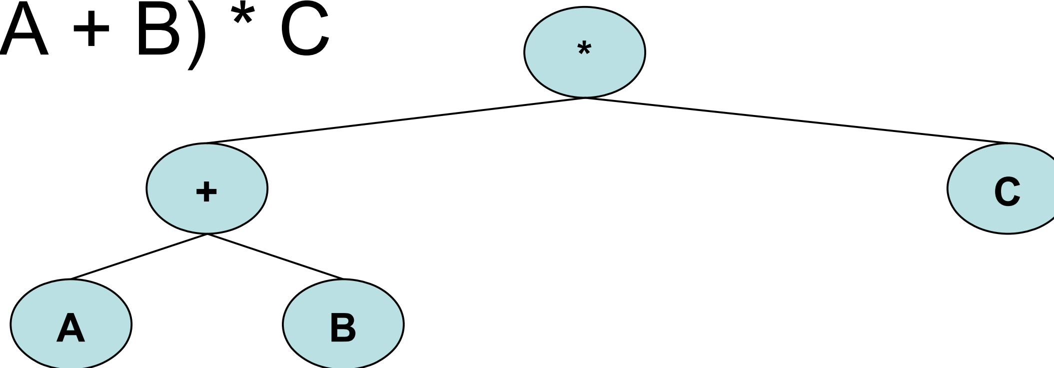
Árvores Binárias – Aplicações

Vejamos alguns exemplos:

A + B * C



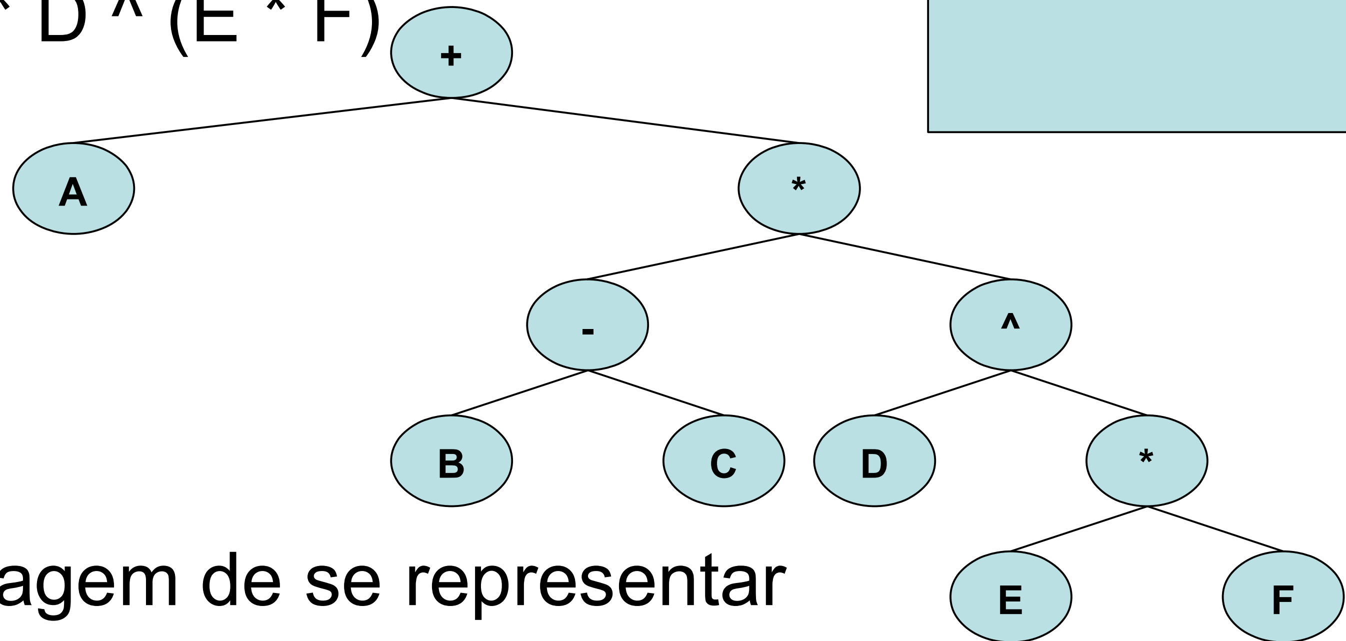
(A + B) * C



UNIVASF

Árvores Binárias – Aplicações

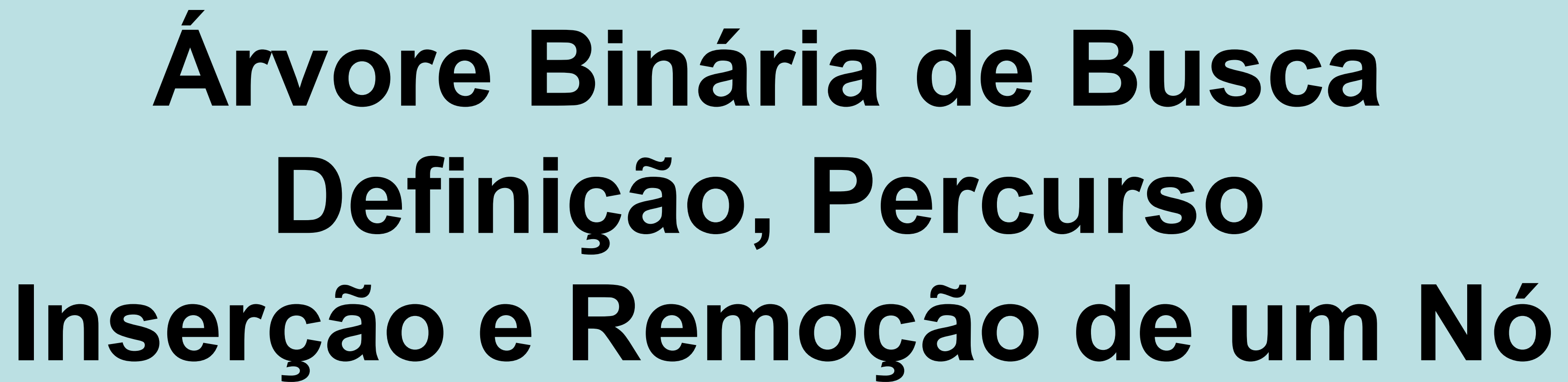
A + (B - C) * D ^ (E * F)



Qual a vantagem de se representar uma expressão assim?

Veremos em breve, quando estudarmos formas de percurso em árvores.

UNIVASF



UNIVASF

Árvore Binária de Busca

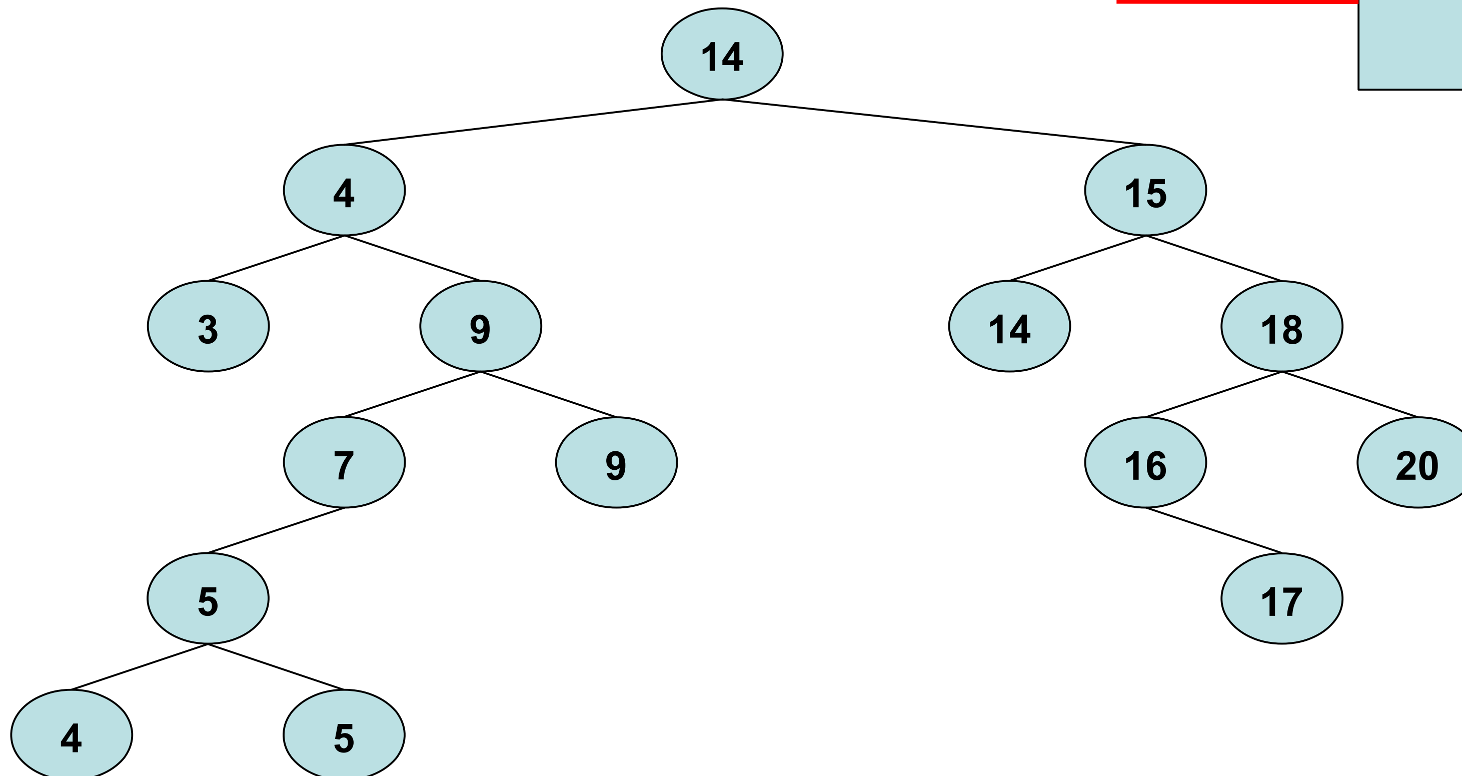
Veremos agora a definição de uma árvore binária de busca.

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.

UNIVASF

Árvore Binária de Busca

Exemplo de uma árvore binária de busca.



Qual seria sua aplicação?

UNIVASF

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.

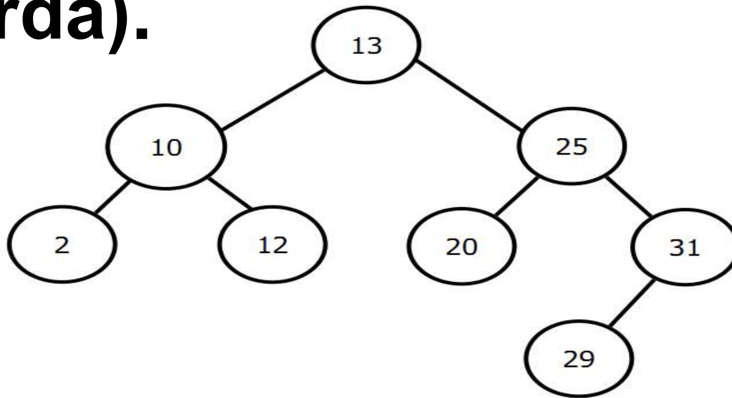
Percurso em largura e em profundidade.

UNIVASF

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

UNIVASF

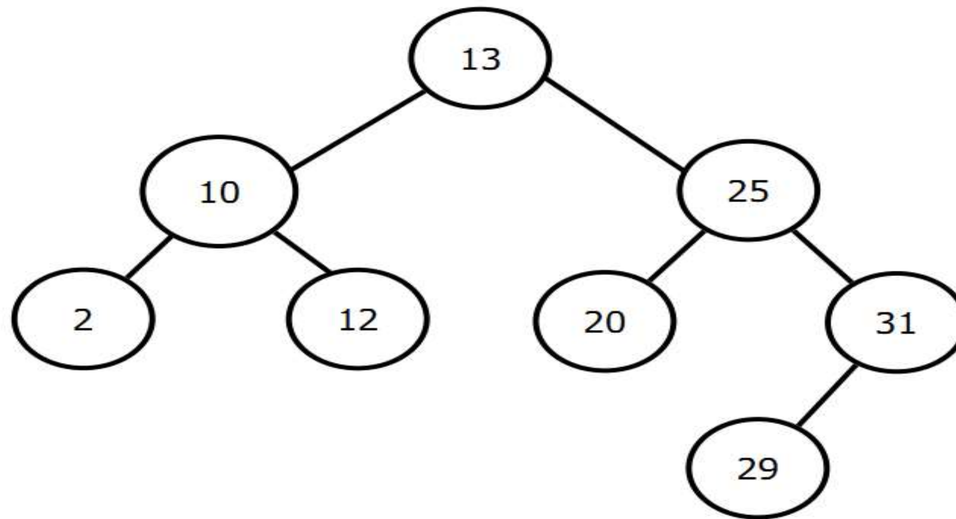
Exercício:

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

```c
void precursoEmLargura(ARV_BIN_ENC arvore)
{
    FILA_ENC fila; /*FILA_ENC armazena ARV_BIN_ENCs*/
    cria_fila(&fila);
    if (arvore)
        ins_fila (fila, arvore);
    while (!eh_vazia_fila(fila))
    {
        printf("%d ", info(cons_fila(fila))); /*Visita ao nó*/
        if (left(cons_fila(fila)))
            ins_fila (fila, left(cons_fila(fila)));
        if (right(cons_fila(fila)))
            ins_fila (fila, right(cons_fila (fila)));
        ret_fila(fila);
    }
}
```

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)

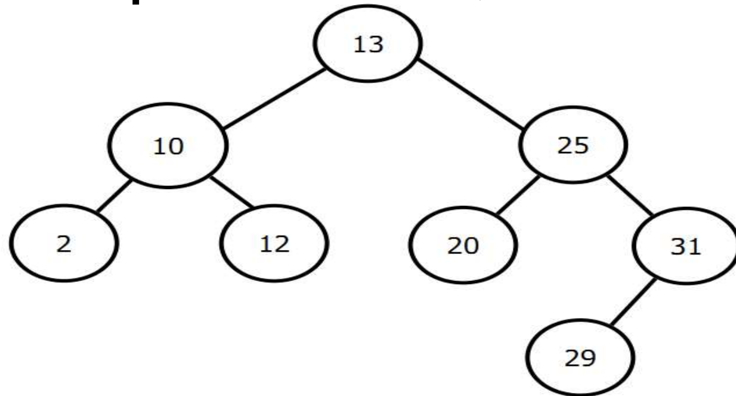
LVR – cruzamento de árvore em in-ordem (conhecido também como ordem simétrica)

LRV – cruzamento de árvore em pós-ordem

UNIVASF

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 explicita (implementada pelo programador) ou utilizando o algoritmo idealizado por Joseph M. Morris.

UNIVASF

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.

UNIVASF

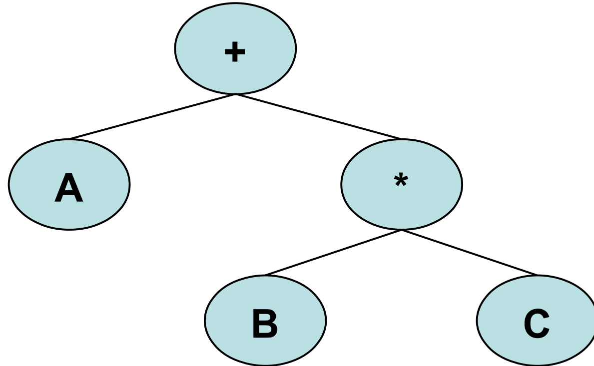
```c
void precursoPreOrdem(ARV_BIN_ENC arvore)
{
    if (arvore)
    {
        printf("%d ", info(arvore));      /*V*/
        precursoPreOrdem(left(arvore));   /*L*/
        precursoPreOrdem(right(arvore)); /*R*/
    }
}
```

```c
void precursoInOrdem(ARV_BIN_ENC arvore)
{
    if (arvore)
    {
        precursoInOrdem(left(arvore));    /*L*/
        printf("%d ", info(arvore));      /*V*/
        precursoInOrdem(right(arvore));   /*R*/
    }
}
```

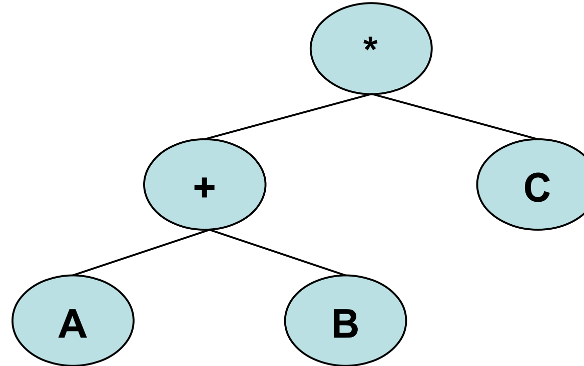
```c
1  void precursoPosOrdem(ARV_BIN_ENC arvore)
2  {
3      if (arvore)
4      {
5          precursoPosOrdem(left(arvore));  /*L*/
6          precursoPosOrdem(right(arvore)); /*R*/
7          printf("%d ", info(arvore));     /*V*/
8      }
9  }
```

Árvore Binária de Busca – Modos de Travessia

Para finalizarmos nossa análise sobre este tema verifique como ficaria o percurso das árvores abaixo em pré-ordem, in-ordem e pós-ordem?



Pré-ordem
+A*BC
In-ordem
A+B*C
Pós-ordem
ABC*+



Pré-ordem
***+ABC**
In-ordem
A+B*C
Pós-ordem
AB+C*

Árvore Binária de Busca

Vimos a seguinte definição para o TAD ARV_BIN_ENC

```
typedef struct node
{
    int info;
    struct node *left;
    struct node *right;
    struct node *father;
} NODE;
typedef NODE * ARV_BIN_ENC;
```

e implementados as operações: **maketree, setleft, setright, info, left, right, father, brother, isleft** e **isright.**

Árvore Binária de Busca

Considerando a mesma definição, do TAD ARV_BIN_ENC, para o TAD ARV_BIN_BUSCA, inclusive suas operações, implemente a operação de inserção de um elemento em uma árvore binária de busca. A qual terá o seguinte protótipo:

void ins_ele (ARV_BIN_BUSCA *arv, int v);

onde **arv** é uma referência para uma referência para a raiz de uma árvore binária de busca qualquer, que inclusive pode ser vazia, e **v** o valor a ser inserido.

UNIVASF

```c
void ins_ele (ARV_BIN_BUSCA *arv, int v) {
    if (!(*arv))
        maketree(arv, v);
    else {
        ARV_BIN_BUSCA father=*arv;
        do {
            if(v<father->info){
                if(father->left)
                    father= father->left;
                else{
                    setleft(father, v);
                    break;
                }
            }
            else{
                if(father->right)
                    father= father->right;
                else{
                    setright(father, v);
                    break;
                }
            }
        }while(1);
    }
}
```

Árvore Binária de Busca

A remoção de um nó em uma árvore binária de busca já não é tão trivial.

A que é proporcional a complexidade do algoritmo de remoção?

Ao número de filhos que o nó a ser removido possui.

Cite as possibilidades.

Nó sem filho – o ponteiro correspondente a seu ascendente é ajustado para nulo e a memória ocupada pelo nó é liberada.

UNIVASF

Árvore Binária de Busca

Nó com um filho – o ponteiro correspondente a seu ascendente é ajustado para apontar para o filho do nó e a memória ocupada pelo nó é liberada.

Nó com dois filhos – para este caso nenhuma operação de apenas uma etapa pode ser executada, discutiremos duas soluções para este caso.

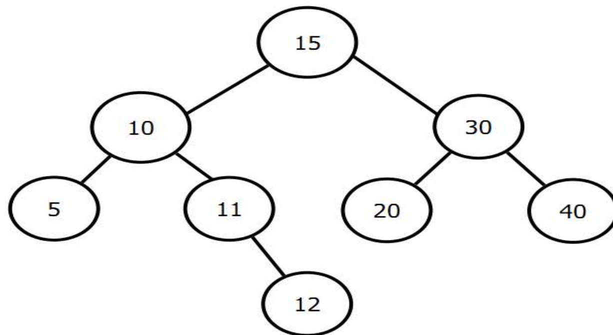
Remoção por fusão e remoção por cópia.

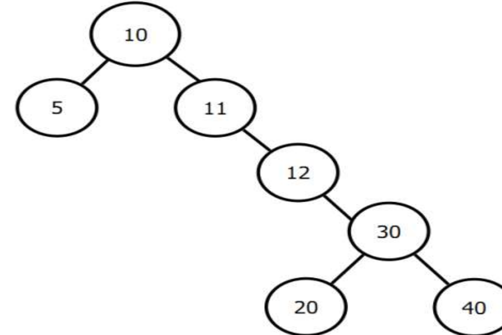
Na remoção por fusão, uma das duas subárvores do nó é extraída e anexada à outra subárvore.

UNIVASF

Árvore Binária de Busca

Para uma melhor compreensão do processo analisaremos a árvore abaixo considerando a remoção do nó com valor 15.





UNIVASF

Árvore Binária de Busca

Com base no que foi discutido codifique uma função, na linguagem C, que implemente a remoção por fusão.

UNIVASF

Árvore Binária de Busca

Outra solução é a Remoção por cópia.

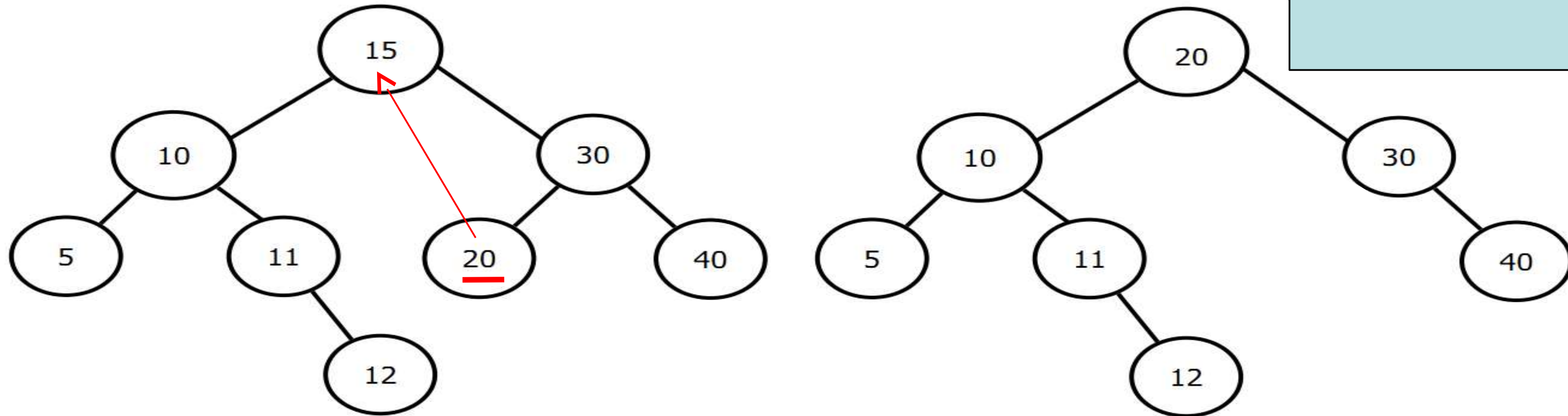
Proposta por Thomas Hibbard e Donald Knuth, propõe que um nó com dois filhos a ser removido pode ser reduzido a uma das duas situações básicas: nó com apenas um filho e nó sem nenhum filho.

Isso é feito substituindo pela chave de seu sucessor imediato a chave que está sendo removida e em seguida removendo o nó que continha a chave do sucessor imediato.

Obs.: o sucessor imediato de um nó é o nó mais à esquerda em sua subárvore à direita.

Árvore Binária de Busca

Vejamos um exemplo:



Exercício:

Com base no que foi discutido codifique uma função, na linguagem C, que implemente a remoção por cópia.