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.

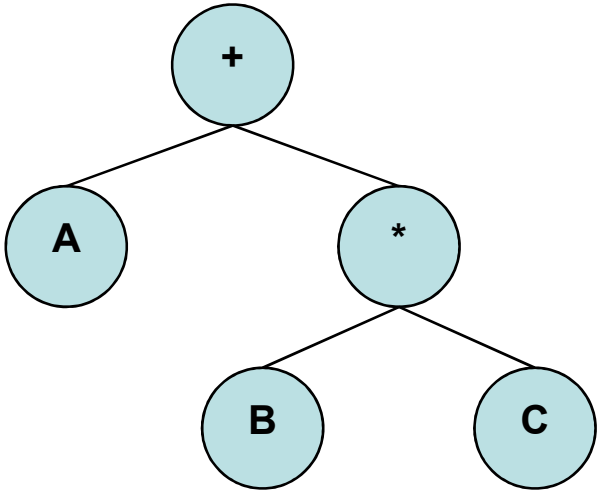
```
void precursoPreOrdem(ARV_BIN_ENC arvore)
{
    if (arvore)
    {
        printf("%d ", info(arvore)); /*V*/
        precursoPreOrdem(left(arvore)); /*L*/
        precursoPreOrdem(right(arvore)); /*R*/
    }
}
```

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

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

Árvore binária de busca – Modos de travessia

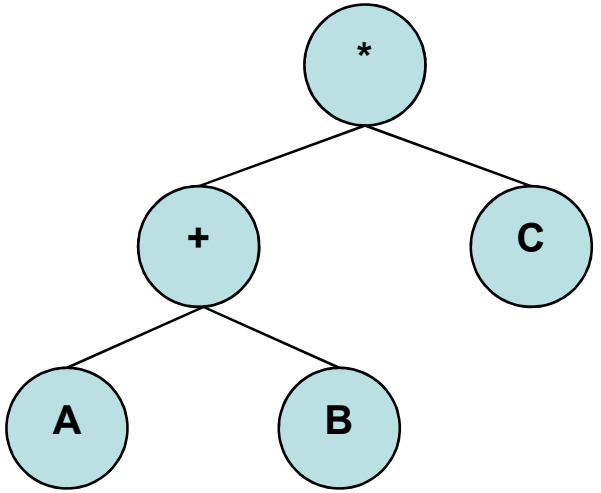
Para finalizarmos nossa análise sobre este tema verifique como ficaria o percurso das árvores abaixo em pré-ordem, in-ordem e pós-ordem?



Pré-ordem
+A*BC

In-ordem
A+B*C

Pós-ordem
ABC*+



Pré-ordem
***+ABC**

In-ordem
A+B*C

Pós-ordem
AB+C*

Assim fica clara a utilidade de se representar uma expressão aritmética como uma árvore.

Porém, é importante salientar que o percurso in-ordem obtém a expressão infixada, contudo, obedecendo apenas à precedência dos operadores, ignorando a capacidade da árvore de representar a utilização de parênteses.

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

```
void ins_ele (ARV_BIN_BUSCA *arv, int v) {
   if (!(*arv))
        maketree(arv, v);
   else {
     ARV_BIN_BUSCA father=*arv;
     do {
       if (v<info(father)){ /*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.

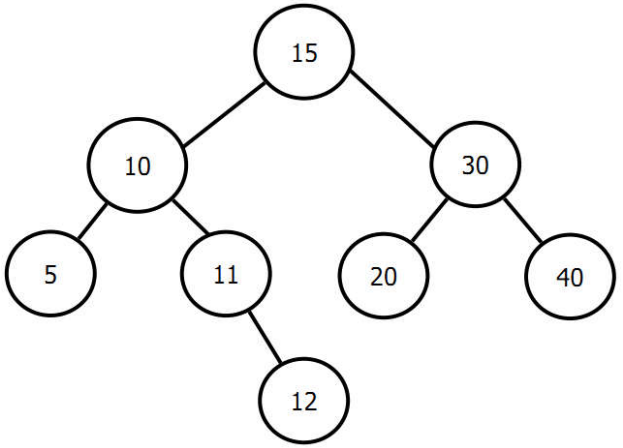
Nó com um filho – o ponteiro correspondente a seu ascendente é ajustado para apontar para o filho do nó e a memória ocupada pelo nó é liberada.

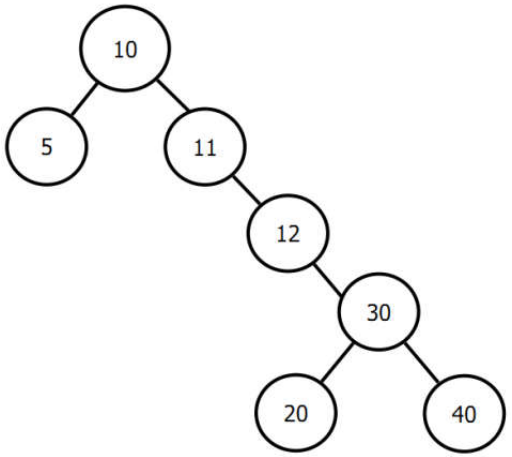
Árvore binária de busca

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.

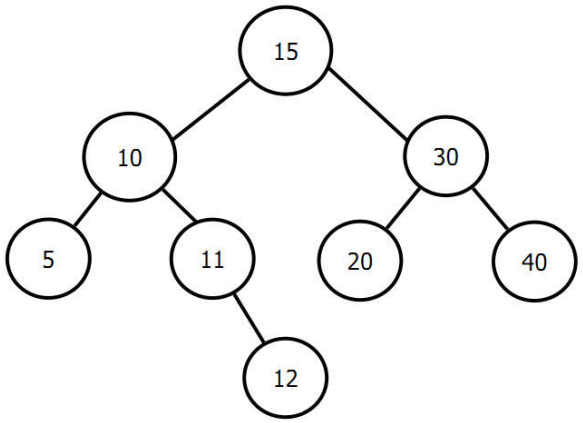
Na remoção por fusão, uma das duas subárvores do nó é extraída e anexada à outra subárvore. Para uma melhor compreensão do processo analisaremos a árvore abaixo considerando a remoção do nó com valor 15.





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

```
void remocaoPorFusao(ARV_BIN_BUSCA *arvore) {
    if (*arvore) {
        ARV_BIN_BUSCA tmp = *arvore;
        if (!((*arvore)->right))
            *arvore = (*arvore)->left;
        else
            if ((*arvore)->left == NULL)
                *arvore = (*arvore)->right;
            else {
                tmp = (*arvore)->left;
                while (tmp->right)
                    tmp = tmp->right;
                tmp->right = (*arvore)->right;
                tmp->right ->father= tmp;
                tmp = *arvore;
                *arvore = (*arvore)->left;
            }
        free (tmp);
    } }
```



Árvore binária de busca

Outra solução é a Remoção por cópia.

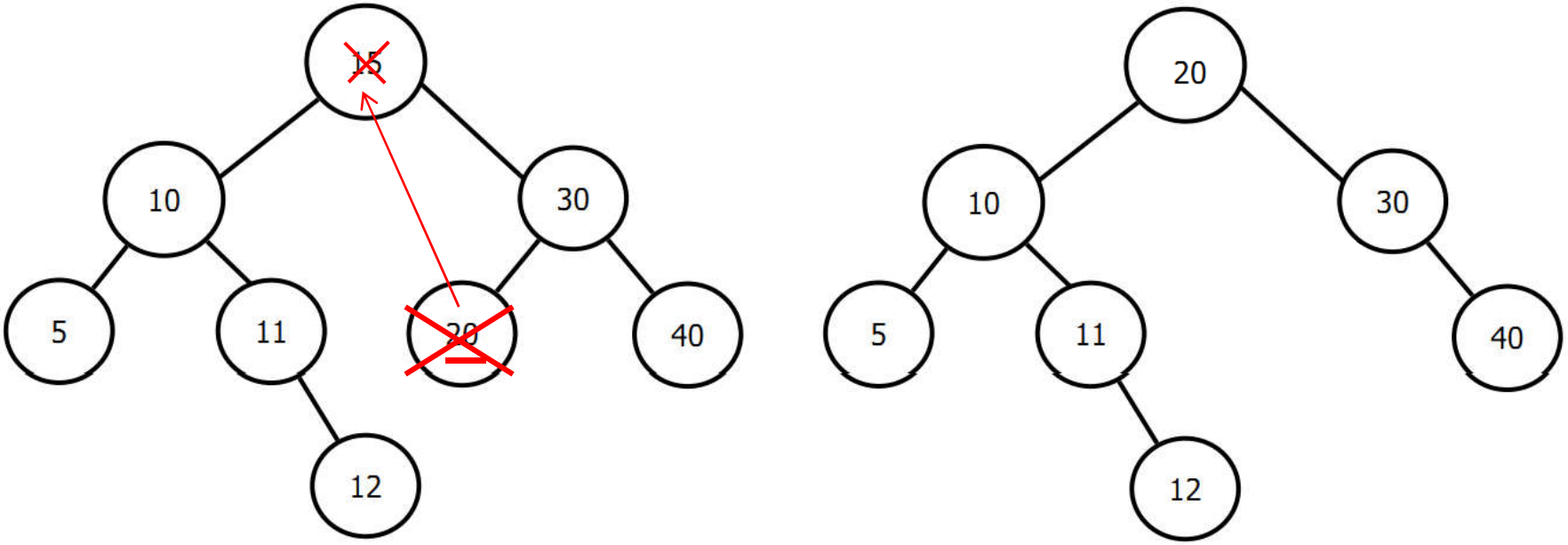
Proposta por Thomas Hibbard e Donald Knuth, propõe que um nó com dois filhos a ser removido pode ser reduzido a uma das duas situações básicas: nó com apenas um filho e nó sem nenhum filho.

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

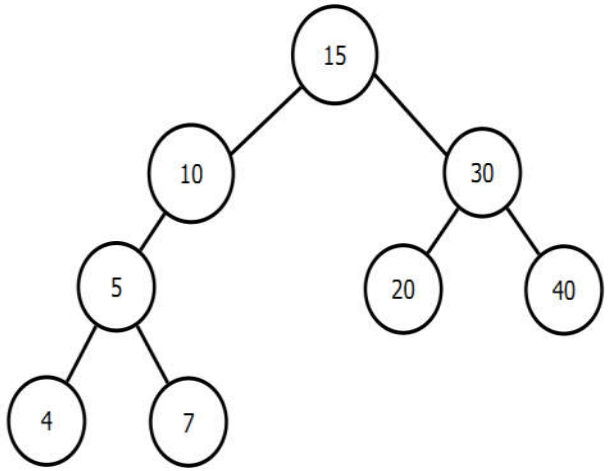
Vejamos um exemplo:



Exercício:

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

```
void remocaoPorCopia(ARV_BIN_BUSCA *arvore) {
    if (*arvore) {
        ARV_BIN_BUSCA tmp = *arvore;
        if ((*arvore)->right == NULL){
            *arvore = (*arvore)->left;
            (*arvore)->father=NULL;
        }else
            if ((*arvore)->left == NULL){
                *arvore = (*arvore)->right;
                (*arvore)->father=NULL;
            }else{
                tmp = (*arvore)->right;
                while (tmp->left!=NULL)
                    tmp = tmp->left;
                 (*arvore)->info = tmp->info;
                if (tmp->father==*arvore){
                    tmp->father->right = tmp->right;
                    tmp->father->right->father = tmp->father;
                }else{
                    tmp->father->left = tmp->right;
                    tmp->father->left->father = tmp->father;
                }
            }
        free (tmp); } }
```



Árvore Balanceada

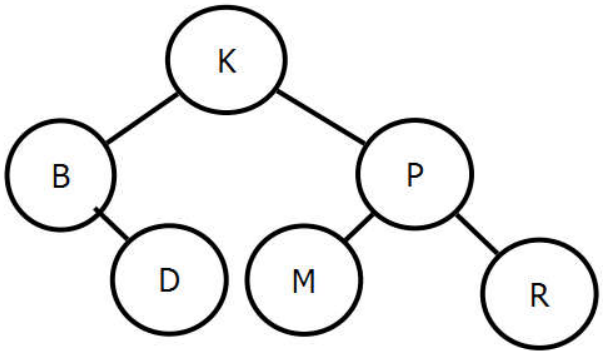
Além das árvores serem muito apropriadas para representar a estrutura hierárquica de um certo domínio, o processo de busca por um elemento em uma árvore tende a ser muito mais rápido do que em uma lista encadeada.

Contudo, para que efetivamente uma busca por um determinado elemento em uma árvore seja eficiente esta, além de ser uma árvore binária de busca, deve ter seus nós adequadamente distribuídos.

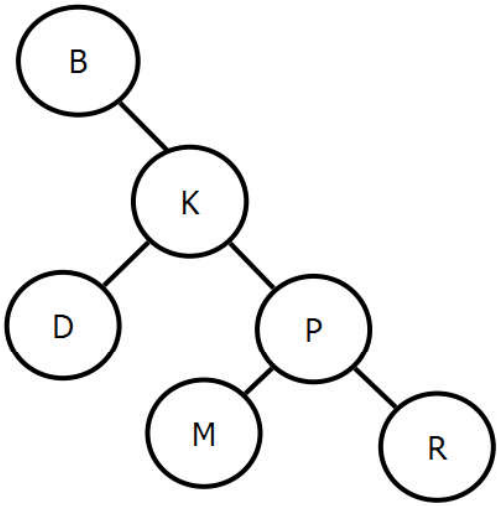
UNIVASF

Árvore Balanceada

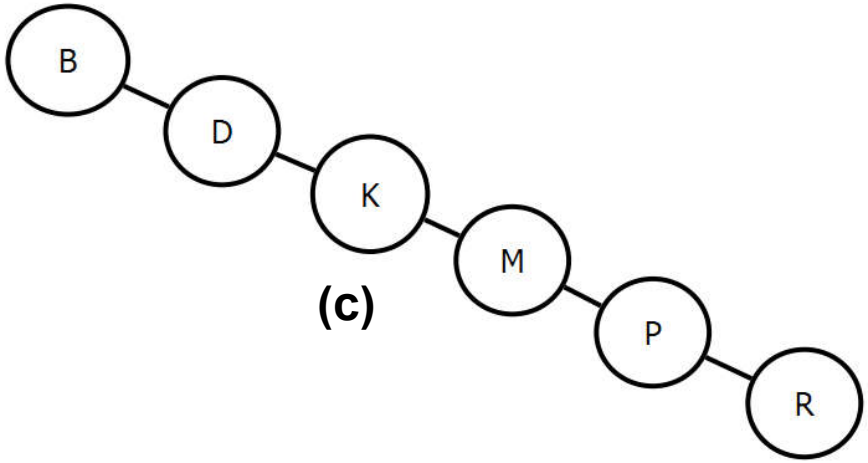
Observe as árvores a seguir:



(a)



(b)



(c)

Árvore Balanceada

Com uma análise das árvores apresentadas podemos concluir que uma grande gama de configurações de árvores binárias de busca é passível de ser obtida com um mesmo conjunto de nós e que o pior caso de uma busca por um nó está diretamente relacionado com a altura da árvore.

Sendo assim, podemos estabelecer a seguinte definição: Uma árvore binária é ***balanceada em altura*** ou simplesmente ***balanceada*** se a diferença na altura de ambas as subárvores de qualquer nó na árvore é zero ou um.

UNIVASF

Árvore Balanceada

Uma definição complementar é a de árvore ***perfeitamente balanceada***.

Uma árvore binária é ***perfeitamente balanceada*** quando, além de ser balanceada, todas as suas folhas encontram-se em um ou dois níveis.

Para facilitar a visualização, uma árvore que possui 10.000 nós pode ser configurada em uma árvore com altura igual a $log_2(10000) = ceil(13.289) = 14$. Ou seja, qualquer elemento é passível de ser localizado com no máximo 14 comparações se a árvore for perfeitamente balanceada.

UNIVASF

Árvore Balanceada

Neste ponto cabe a seguinte pergunta: Ao se analisar uma árvore pode-se determinar qual dentre os seus nós seria uma raiz adequada para torná-la perfeitamente balanceada? Qual seria este nó?

O nó cuja sua chave (valor) representa a mediana das chaves presentes nos nós que compõem a árvore, para uma árvore com número impar de nós. Ou o nó cuja sua chave (valor) representa um dentre os dois valores mais próximos da mediana das chaves presentes nos nós que compõem a árvore, para uma árvore com número par de nós.

Uma estratégia baseada nesta observação que pode ser utilizada para balancear uma árvore é:

Árvore Balanceada

- Retire os dados da árvore e armazene-os em um vetor;

- Após todos os dados terem sido armazenados no vetor, ordene-o;

- Agora determine como raiz o elemento do meio do vetor;

- O vetor consistirá agora em dois subvetores. O filho esquerdo da raiz será o nó com valor no meio do subvetor constituído do início do vetor até o elemento escolhido como raiz;

- Um procedimento similar é adotado para a definição do filho direito da raiz;

- Este processo se repete até não existirem mais elementos a serem retirados do vetor.

Exercício: Determine o código fonte de uma função, na linguagem C, que implementa este processo.

UNIVASF