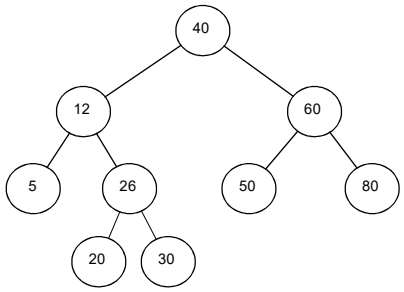
PUC-Rio – Estruturas de Dados – INF1010
Simulado P1 – 2018_2 (baseado em provas anteriores dos Profs. Casanova e Noemi)

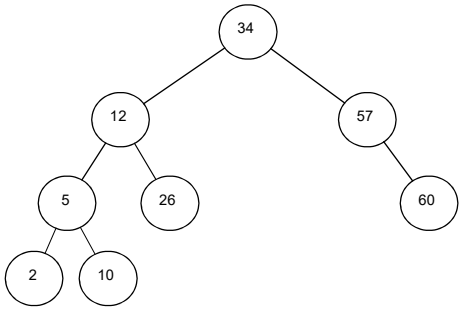
1. Suponha que queremos escrever uma função que retorne o dado associado a uma chave em um nó de uma árvore binária. Considerando a representação de árvores binárias usada nos laboratórios, analise a função busca mostrada a seguir.

```
int busca(Nodo* m, int c) {
   while (m != NULL && m->chave > c) m = m->esq;
   while (m != NULL && m->chave < c) m = m->dir;
   if (m==NULL) return -1;
     else return m->dados;
}
```

Explique por que a função não é correta, dando um exemplo de chave que não seria encontrada na árvore binária a seguir, apesar de estar nela. Diga o que a função retornaria para essa chave, sabendo que o valor do dado é sempre igual ao da chave.



2. Suponha a árvore AVL mostrada a seguir.



Como ficará a árvore se realizarmos cada uma das operações (*atenção:* cada uma das operações indicadas ocorre na árvore original mostrada, uma após a outra!). Desenhe a nova árvore para cada caso. Indique o nó desregulado e o tipo de rotação efetuada.

(a) inserção com chave 65

(b) inserção com chave 1

(c) inserção com chave 11

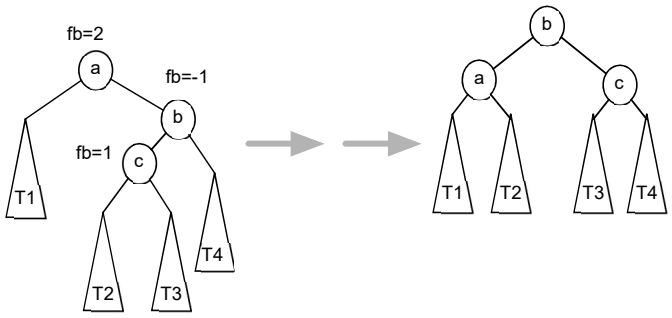
(d) remoção da chave 57

3. Considere a função eh_abb (vista em laboratório), que determina se uma determinada árvore binária é uma árvore binária de busca, isto é, se para cada nó *n* da árvore, todas as chaves da subárvore à sua esquerda são menores que a chave de *n*, e todas as chaves na sua subárvore à direita são maiores que a chave de *n*. Uma outra forma de implementar a função, diferente da discutida em sala, é escrever uma função que recebe dois parâmetros, min e max, que indicam os valores mínimo e máximo permitidos na subárvore a ser visitada a cada chamada recursiva. Escreva a função eh_abbrecursiva que implementa essa ideia. Considere a representação de árvore binária usada nos laboratórios.

```
static int eh_abbrecursiva (Nodo * m, int min, int max) {
 ...
}

int eh_abb (Nodo*m)
{
      return ( eh_abbrecursiva (m, INT_MIN, INT_MAX) );
      /* INT_MIN e INT_MAX são, respectivamente, o menor e maior inteiro representáveis na
       máquina utilizada. */
}
```

4. Considere a operação de rotação dupla mostrada abaixo, para correção de árvore AVL desbalanceada. Diga quais serão os fatores de balanceamento finais dos nós a, b e c, *explicando* como podem ser calculados a partir da configuração inicial.



5. Escreva uma função *mostracaminholongo* que imprima todas as chaves em nós no caminho mais longo da raiz até uma folha de uma árvore AVL. Se existirem vários caminhos de mesmo tamanho, a função pode imprimir qualquer um deles. Suponha a representação de árvore usada nos laboratórios. Sua função pode ser recursiva ou não.

```
void mostracaminholongo (Nodo *m);
```

6. Qual a ordem da complexidade em tempo dos algoritmos A e B abaixo? Explique sua resposta.

A-
```
int fat (int n) {
        if (n==0)
                return 1;
        return n* fat (n-1);
  }
```

B-
```
for ( i=1; i < n; i *= 2 ) {
        for ( j = n; j > 0; j /= 2 ) {
                for ( k = j; k < n; k += 2 ) {
                        sum += (-j * k) << i/2;
                }
        }
}
```

7. Considere o problema de organizar um conjunto de alunos de tal forma que os alunos com maior CR tenham prioridade sobre os alunos que tem menor CR. Caso dois alunos tenham o mesmo CR, a ordem é indistinta. Assuma ainda que se deseja realizar as seguintes operações:

   - Organizar inicialmente o conjunto de alunos, segundo a prioridade por CR;
   - Remover o aluno com maior CR;
   - Incluir um novo aluno, segundo a prioridade por CR.

   a) Porque um *heap* seria a estrutura mais eficiente para resolver o problema acima? Explique sua resposta.

   b) Considere inicialmente um conjunto de alunos com os seguintes CRs: 6.2, 7.2, 7.1, 6.0, 7.1. Mostre como construir o *heap*, utilizando um vetor, da forma mais eficiente possível, para organizar este conjunto de alunos com prioridades dadas pelo CR.

   c) Escreva em C uma função que recebe como argumento um *heap*, organizado como nos itens anteriores, e remove do *heap* o aluno com o *segundo* maior CR, usando o menor espaço e o menor tempo possível. A rotina deve retornar 1 se a alteração ocorreu corretamente e 0 em caso contrário. Assuma que o *heap* seja definido como abaixo:

```
struct _heap {
  int max;              /* tamanho máximo do heap          */
  int pos;              /* próxima posicao disponível no vetor */
  float* prioridade;    /* vetor das prioridades           */
};

typedef struct _heap Heap;
```

   Assuma ainda que a assinatura da rotina seja:

```
int heap_altera(Heap* heap)
```

   A rotina não deve chamar as operações de inserção ou remoção de um heap apresentadas em sala, mas poderá usar uma das operações auxiliares apresentadas em sala.