Relatório de implementação da AVL

- Implementação:
  - A AVL foi implementada de forma que, primeiro se faz as operações de inclusão e remoção como se fosse uma BST normal, e após esse passo se faz o balanceamento da árvore.
  - O código utiliza scanf para receber os parâmetros que irão definir a operação(**r** para remoção e **i** para inclusão) e para ler o inteiro que a chave irá receber. Isso baseando-se em que a entrada padrão é redirecionada para um arquivo .txt e a saída também;
  - Cada nodo foi implementado com struct, no qual seus elementos são:
    - Um inteiro que irá guardar a altura do nodo;
    - Um inteiro que irá guardar a chave;
    - Um ponteiro para o pai;
    - Um ponteiro para o filho da esquerda;
    - Um ponteiro para o filho da direita;
- Funções utilizadas:
  - **incluiNodo(nodo *T, int chave):**
    - Recebe um ponteiro para raiz e um inteiro que será a chave. Se a árvore está vazia (T == NULL) a função cria o primeiro nodo, se não ela irá aplicar uma busca pela chave e chamará uma função que insere o nodo. Por fim chama a função que efetua o balanceamento à partir do nodo inserido.
  - **insereNodo(nodo *p, int chave):**
    - Função que insere um nodo na posição desejada.
  - **buscaEx(nodo *T, int chave):**
    - Busca utilizada na função incluiNodo. Retorna o nodo pai do nodo que será inserido futuramente.
  - **excluiNodo(nodo *T, int chave):**
    - Efetua a exclusão pelo antecessor, analisando todos os casos possíveis numa inclusão em BST e fazendo a ligação de todos os ponteiros necessários. Quando um caso é reconhecido, as devidas providências são tomadas e a função é encerrada chamando a função de balanceamento e retornando seu valor para main.
  - **busca(nodo *T, int chave):**
    - Função de busca utilizada em excluiNodo. Retorna o nodo que se deseja excluir, ou retorna NULL quando não é encontrado. Esse algoritmo foi implementado de acordo com o visto em sala.
  - **rotacaoDireita(nodo *x):**
    - Efetua a rotação a direita de um nodo. O nodo que vai para baixo é passado como parâmetro. Após isso, re-calcula a altura dos nodos que sofreram a rotação.

- **rotacaoEsquerda(nodo *x):**
  - Efetua a rotação para a esquerda de um nodo. O nodo que vai para baixo é passado como parâmetro. Após isso, re-calcula a altura dos nodos que sofreram a rotação.
- **balanceia(nodo *p):**
  - A função confere se o nodo está balanceado e aplica as rotações para cada caso. Começa conferindo pelo nodo passado como parâmetro e sobe até a raiz, sempre fazendo as rotações necessárias e re-calculando a altura de cada nodo que passa. Aplica as rotações de acordo com os casos de zig-zig ou zig-zag. Se a diferença entre as alturas dos nodos filhos do que está sendo avaliado forem iguais a 2 ou -2, está desbalanceado e o nodo que está sendo conferido é o nodo problema. Após isso, a função vê qual é o maior caminho e utiliza a função **fator** para decidir se é caso zig-zig ou zig-zag. Após esse processo, utiliza-se um ponteiro que sobe até a raiz e retorna ele.
- **calculaAltura(nodo *x):**
  - Calcula o tamanho do maior caminho do nodo parâmetro até uma folha. Essa medida é a altura guardada em cada nodo.
- **imprimeAVL(nodo *T):**
  - Algoritmo que imprime os nodos em ordem, imprimindo também o nível de cada nodo de acordo com o especificado no trabalho. O algoritmo aqui implementado é baseado no que foi passado em sala de impressão em ordem.
- **contaNivel(nodo *T):**
  - Calcula o nível do nodo parâmetro. Essa função é utilizada na impressão somente. O cálculo se baseia num laço que vai do nodo até a raiz.
- **fator(nodo *p):**
  - Retorna a diferença entre os nodos da esquerda e da direita dos nodos avaliados. Se a diferença é -1 o maior caminho é o da direita, se for 1 é o da esquerda. Retorna 0 se o nodo parâmetro for NULL. Essa função é utilizada para decidir o caso no balanceamento.