1. Node: Esta estructura interna representa un nodo en el árbol AVL. Tiene un campo key para almacenar el valor, punteros left y right para apuntar a los hijos izquierdo y derecho respectivamente, y un campo height que almacena la altura del nodo.

2. height(Node* node): Esta función auxiliar calcula la altura de un nodo dado. Si el nodo es nulo, se devuelve 0. De lo contrario, se devuelve el valor del campo height del nodo.

3. balanceFactor(Node* node): Esta función auxiliar calcula el factor de equilibrio de un nodo dado. El factor de equilibrio se define como la diferencia entre la altura del subárbol izquierdo y la altura del subárbol derecho. Si el nodo es nulo, se devuelve 0.

4. updateHeight(Node* node): Esta función auxiliar actualiza la altura de un nodo en función de las alturas de sus hijos. Se utiliza después de realizar una rotación o inserción/remoción para garantizar que la altura del nodo sea correcta.

5. rotateRight(Node* node): Esta función realiza una rotación hacia la derecha en el nodo dado. Se utiliza para restaurar el equilibrio en caso de desequilibrio hacia la izquierda.

6. rotateLeft(Node* node): Esta función realiza una rotación hacia la izquierda en el nodo dado. Se utiliza para restaurar el equilibrio en caso de desequilibrio hacia la derecha.

7. balance(Node* node): Esta función comprueba y restaura el equilibrio en el nodo dado y sus subárboles. Se utiliza después de insertar o eliminar un nodo.

8. insert(Node* node, const T& key): Esta función inserta un nuevo nodo con la clave dada en el subárbol con raíz en el nodo dado. Utiliza recursión para encontrar la ubicación correcta para la inserción y luego llama a la función balance para restaurar el equilibrio.

9. findMin(Node* node): Esta función encuentra el nodo con el valor mínimo en el subárbol con raíz en el nodo dado. Se utiliza en la eliminación de nodos.

10. remove(Node* node, const T& key): Esta función elimina un nodo con la clave dada del subárbol con raíz en el nodo dado. Utiliza recursión para encontrar el nodo a eliminar y luego realiza las acciones necesarias para mantener el árbol equilibrado. También actualiza la altura del nodo y realiza rotaciones si es necesario.

11. inorderTraversal(Node* node): Esta función realiza un recorrido en orden (izquierda, raíz, derecha) en el subárbol con raíz en el nodo dado. Imprime los valores de los nodos en orden creciente.

12. preorderTraversal(Node* node): Esta función realiza un recorrido en preorden (raíz, izquierda, derecha) en el subárbol con raíz en el nodo dado. Imprime los valores de los nodos en el orden en que se visitan.

13. printGraphicalUtil(Node* node, int level): Esta función imprime una representación gráfica vertical del árbol AVL. Utiliza una variación del recorrido en orden para imprimir los nodos en el orden correcto y agrega espacios y líneas verticales para mostrar la estructura del árbol.

14. generateDotUtil(std::ofstream& file, Node* node): Esta función auxiliar genera un archivo DOT que representa el árbol AVL en formato de grafo. Utiliza el formato DOT para especificar los nodos y las conexiones entre ellos.

15. getNodeCount(Node* node): Esta función cuenta y devuelve la cantidad de nodos en el subárbol con raíz en el nodo dado. Utiliza recursión para contar los nodos de los subárboles izquierdo y derecho, y agrega 1 para contar el nodo actual.

16. isBalanced(Node* node): Esta función verifica si el subárbol con raíz en el nodo dado está equilibrado según las reglas de un árbol AVL. Comprueba el factor de equilibrio del nodo y sus subárboles y retorna true si todos cumplen con la condición de equilibrio (-1 ¡= factor de equilibrio ¡= 1).

17. AVL(): El constructor de la clase AVL inicializa el árbol AVL estableciendo el puntero de raíz en nullptr.

18. insert(const T& key): Este método público permite insertar un valor en el árbol AVL. Llama a la función privada insert pasando la raíz del árbol y el valor a insertar.

19. remove(const T& key): Este método público permite eliminar un valor del árbol AVL. Llama a la función privada remove pasando la raíz del árbol y el valor a eliminar.

20. printInorder(): Este método público realiza un recorrido en orden en el árbol AVL e imprime los valores de los nodos en orden creciente.

21. printPreorder(): Este método público realiza un recorrido en preorden en el árbol AVL e imprime los valores de los nodos en el orden en que se visitan.

22. printGraphical(): Este método público imprime una representación gráfica vertical del árbol AVL.

23. generateDotFile(const std::string& filename): Este método público genera un archivo DOT que representa el árbol AVL en formato de grafo. El archivo se guarda con el nombre especificado.

24. generateDotFileAndPNG(const std::string& dotFilename, const std::string& pngFilename): Este método público genera un archivo DOT y un archivo PNG del árbol AVL. El archivo DOT se guarda con el nombre especificado y luego se utiliza el comando dot de GraphViz para convertir el archivo DOT en un archivo PNG.

25. getNodeCount(): Este método público devuelve la cantidad total de nodos en el árbol AVL llamando a la función privada getNodeCount pasando la raíz del árbol.

26. isBalanced(): Este método público verifica si el árbol AVL está equilibrado según las reglas de un árbol AVL llamando a la función privada isBalanced pasando la raíz del árbol.