Diccionario de Scrabble

Realizado por: Pablo Ariza, Alejandro Castro y Santiago Otalora

---

➔ Scrabble:

- Scrabble es un juego de mesa patentado en 1948 y, actualmente producido por las empresas *Hasbro* y *Mattel.* El objetivo del juego es obtener la mayor cantidad de puntos formando palabras en el tablero tanto de forma horizontal como vertical; sin importar si se cruzan entre sí. Más bien parecido a un crucigrama.

- El juego admite entre 2 y 4 jugadores. Se empiezan con 7 fichas con letras impresas y mediante ellas el jugador debe formar una palabra tratando de obtener la mayor cantidad de puntos. Este proceso consta en obtener las distintas combinaciones de letras para obtener palabras validas en el idioma en el que se está jugando. Para el caso original del juego el idioma es ingles.

- Ya que cada letra cuenta con su propio puntaje, se han desarrollado herramientas de apoyo conocidas como constructores de palabras de Scrabble. Cada uno tiene distintas funcionalidades como por ejemplo obtener las posibilidades de palabra a construir.

- Para el desarrollo de este proyecto se realizó la primera aproximación a lo que será una herramienta de este estilo. La carga de las distintas palabras del idioma inglés a partir de un archivo tanto de forma normal como de forma inversa y la adquisición del puntaje por palabra fueron los puntos a tener en cuenta durante el desarrollo de este componente.

- Como ya se dijo anteriormente el componente se divide en carga de palabras y obtener el puntaje de una palabra. Para el primer caso, se realizaron dos comandos, *init* e *init_inverse.* La diferencia entre estos dos comando es la forma como se cargan las palabras dentro del programa. Para el primer caso se cargan de derecha a izquierda y para el segundo se cargan de forma inversa. El segundo elemento del componente se resume en obtener el valor de una palabra válida y que se encuentre dentro del diccionario cargado.

- Ahora bien, durante este documento se explicarán los distintos análisis e implementaciones que se desarrollaron para llevar a cabo con éxito los objetivos propuestos para este componente.

❖ Entradas:

- Primeramente se examinó las distintas entradas que puede haber en el programa. A partir de esto se determinaron las siguientes:
  - El principal objeto de interacción del usuario con el sistema es el ingreso de una cadena de caracteres por parte del usuario. Este comando ingresado por el usuario determina la acción que realizará el sistema.
  - Un archivo que contiene un diccionario de palabras aceptadas en el idioma inglés.

❖ Salidas:

- Para el caso de las respuestas del sistema para el usuario se determinaron distintas categorías dependiendo de la función que el usuario ejecute sobre el sistema.

  - Función init y función init_inverse:
    - Diccionario ya inicializado: en este caso la salida indica que la función *init* o *init_inverse* ya ha sido ejecutada exitosamente una vez.
    - Archivo no existe: en este caso la salida indica que el archivo que se indicó luego de la función *init* o *init_inverse* no existe.
    - Resultado exitoso: en este caso la salida indica que el diccionario fue cargado exitosamente al sistema.

      Nota: durante la carga de las palabras al diccionario se mostrará errores cuando se quiera ingresar palabras con caracteres inválidos.

  - Función score:
    - Palabra no existe: en este caso la salida indica que la palabra indicada en el comando no existe dentro del diccionario antes creado.
    - Letras inválidas: en este caso la salida indica que la palabra indicada en el comando contiene caracteres inválidos.

        - Resultado exitoso: en este caso la salida indica el puntaje de la palabra indicada en el comando.

    - Función ayuda :
        - Al ingresar únicamente el comando *ayuda* se muestran los distintos comandos soportados por el sistema.
        - Al ingresar el comando *ayuda* junto con el nombre de cualquier otro comando se mostrará lo que realiza dicho comando y como debe de ser ejecutado en el sistema.

    - Función exit:
        - Esta función únicamente termina el programa sin realizar salidas por pantalla.

❖ Procedimiento principal:

➢ El procedimiento principal del programa realiza la captura del comando ingresado por el usuario y determina cuál camino seguir. Además de esto se crean las distintas variables para el funcionamiento del programa.

➢ Para las funciones *init* e *init_inverse* se realiza el llamado a la función auxiliar de lectura del archivo con los parámetros necesarios para su funcionamiento dependiendo de la función requerida. Para la función *score* se llama a la función respectiva y luego se verifica si se encontró el puntaje de la palabra, si la palabra tiene caracteres inválidos o si la palabra no existe en el diccionario.

➢ Para el comando ayuda se realiza el llamado a la función respectiva con un parámetro que indica sobre qué comando se requiere la ayuda. Por último, la función exit únicamente termina el programa en ejecución.

❖ Operaciones auxiliares:

➢ Como ya se dijo anteriormente, el programa principal se apoya en métodos auxiliares a los cuales delega tareas para que la rutina principal no sea tan extensa. Estos métodos son:

- ayuda(comando): este método auxiliar muestra una ayuda por pantalla diferente para cada *comando*. Dado el caso que el comando digitado por el usuario se únicamente la palabra “ayuda” se muestra por pantalla un consolidado de los comandos soportados por el sistema.

- leerArchivo(tree,comando,tipo): este método auxiliar realiza la apertura del archivo y con cada palabra que hay en el archivo llama la función insertarPalabra. Un aspecto importante para recalcar de esta función es la conversión de los caracteres que conforman la palabra a insertar a minúsculas. Esta función imprime en pantalla el error de no apertura del archivo o el éxito en la operación de inicialización del diccionario.
- insertarPalabra(tree, palabra): este método auxiliar inserta la palabra dada en el árbol. Este proceso se realiza mediante la búsqueda del nodo sobre el cual se insertará la palabra; dado que el nodo no se encuentre, se crea un nuevo nodo de la primera letra de la palabra y se inserta dicho nodo en el árbol. En este método de igual manera se imprime por pantalla el error de que la palabra a ingresar tiene caracteres inválidos.
- validarPalabra(palabra): este método auxiliar verifica que la palabra dada no posea símbolos inválidos. Se retorna el número 1 dado que la palabra presente este inconveniente, y se retorna 0 en el caso contrario.
- voltearPalabra(palabra): este método auxiliar invierte la palabra dada. Es decir, dada cualquier cadena de caracteres retorna la misma cadena pero en sentido inverso.
- scorePalabra(tree, palabra): este método auxiliar retorna el puntaje de la palabra dada. Este proceso se realiza mediante la búsqueda primeramente del nodo al cual pertenece la palabra para luego extraer el puntaje de dicha palabra. El método retorna el puntaje de la palabra dada, el número -1 para indicar que la palabra contiene caracteres inválidos o el número -2 para indicar que la palabra no se encuentra dentro del diccionario.

❖ Diseño de los tipos abstractos de datos.

❑ Para poder desarrollar el componente en cuestión, se realizó el diseño de tipos abstractos de datos (TAD) que representarán la organización de los datos dentro del programa en ejecución. A continuación, se presenta en forma de texto cómo están compuestos cada uno de los TAD y, posteriormente, se presenta un gráfico que muestra la relación entre los distintos TAD.

TAD NodoVocabulario:

Conjunto mínimo de datos:

- letra, caracter del abecedario.
- palabras, mapa que usa como llave la palabra válida y como valor el puntaje asignado a esa palabra.

Operaciones:

- NodoVocabulario(), constructor de un nuevo nodo vacío.
- NodoVocabulario(letraNodo), constructor de un nuevo nodo cuya letra tendrá el valor de letraNodo.
- NodoVocabulario(letraNodo,palabrasNodo), constructor de un nuevo nodo con letra igual al valor de letraNodo y palabras igual al mapa palabrasNodo.
- ~NodoVocabulario(), destructor.
- setLetra(letraNodo), asigna el valor de letraNodo a letra.
- setPalabras(palabrasNodo), fija el valor de palabras como palabrasNodo.
- getLetra(), retorna el valor de letra.
- getPalabras(), retorna un mapa igual a palabras.
- insertarPalabra(palabra), inserta la palabra al mapa.
- calcularPuntaje(palabra), calcula el puntaje de una palabra.

TAD BinaryNodeAVL:

Conjunto mínimo de datos:

- data, NodoVocabulario, dato del nodo
- height, entero positivo, altura del nodo
- left,BinaryNodeAVL, apuntador al hijo izquierdo
- right,BinaryNodeAVL, apuntador al hijo derecho

Operaciones:

- BinaryNodeAVL(), constructor de un nuevo nodo vacío.
- BinaryNodeAVL(val), constructor de un nuevo nodo con valor de val.
- BinaryNodeAVL(val, rightSon, leftSon), constructor de un nodo con todos sus datos definidos.
- ~BinaryNodeAVL(), destructor.
- getData(), retorna el dato del nodo.
- getRight(), retorna un apuntador al hijo derecho.
- getLeft(), retorna un apuntador al hijo izquierdo.
- getHeight(), retorna la altura del nodo.
- setData (val), fija el valor del dato.
- setRight (val), fija el valor del hijo derecho.
- setLeft (val), fija el valor del hijo izquierdo.
- setHeight (val), fija el valor de la altura del nodo.
- max(), retorna un apuntador al extremo derecho.
- min(), retorna un apuntador al extremo izquierdo.

- insert (nval), inserta el dato nval en el árbol.
- erase (val), elimina el dato val del árbol.
- search(val), busca el nodo con valor val en el árbol.
- search(val), busca un nodo que tenga el caracter indicado en val dentro de su NodoVocabulario.
- searchFather(val), retorna el nodo padre del nodo con el mismo valor de val.
- nodeHeight(), calcula la altura del nodo.
- updateHeight(), actualiza las alturas de los nodos del árbol
- inOrder(), imprime el recorrido en inOrden del árbol.
- preOrder(), imprime el recorrido en preOrden del árbol.
- posOrder(), imprime el recorrido en posOrden del árbol.
- levelOrder(List), guarda el recorrido por niveles el árbol en un contenedor.
- descendants(), imprime los datos del nodo y el de sus dos hijos, si existen.
- balanceCheck(father, root), retorna true si el árbol está balanceado, si no retorna false.
- rightRotation(), retorna un apuntador al nuevo padre después de realizar la rotación por derecha.
- leftRotation(), retorna un apuntador al nuevo padre después de realizar la rotación por izquierda.
- rightLeftRotation(), retorna un apuntador al nodo después de haber hecho una rotación por izquierda y luego por derecha.
- leftRightRotation(), retorna un apuntador al nodo después de haber hecho una rotación por derecha y luego por izquierda.
- balance(type), balancea el árbol, y retorna un apuntador a la nueva raíz.

TAD BinaryTreeAVL:

Conjunto mínimo de datos:

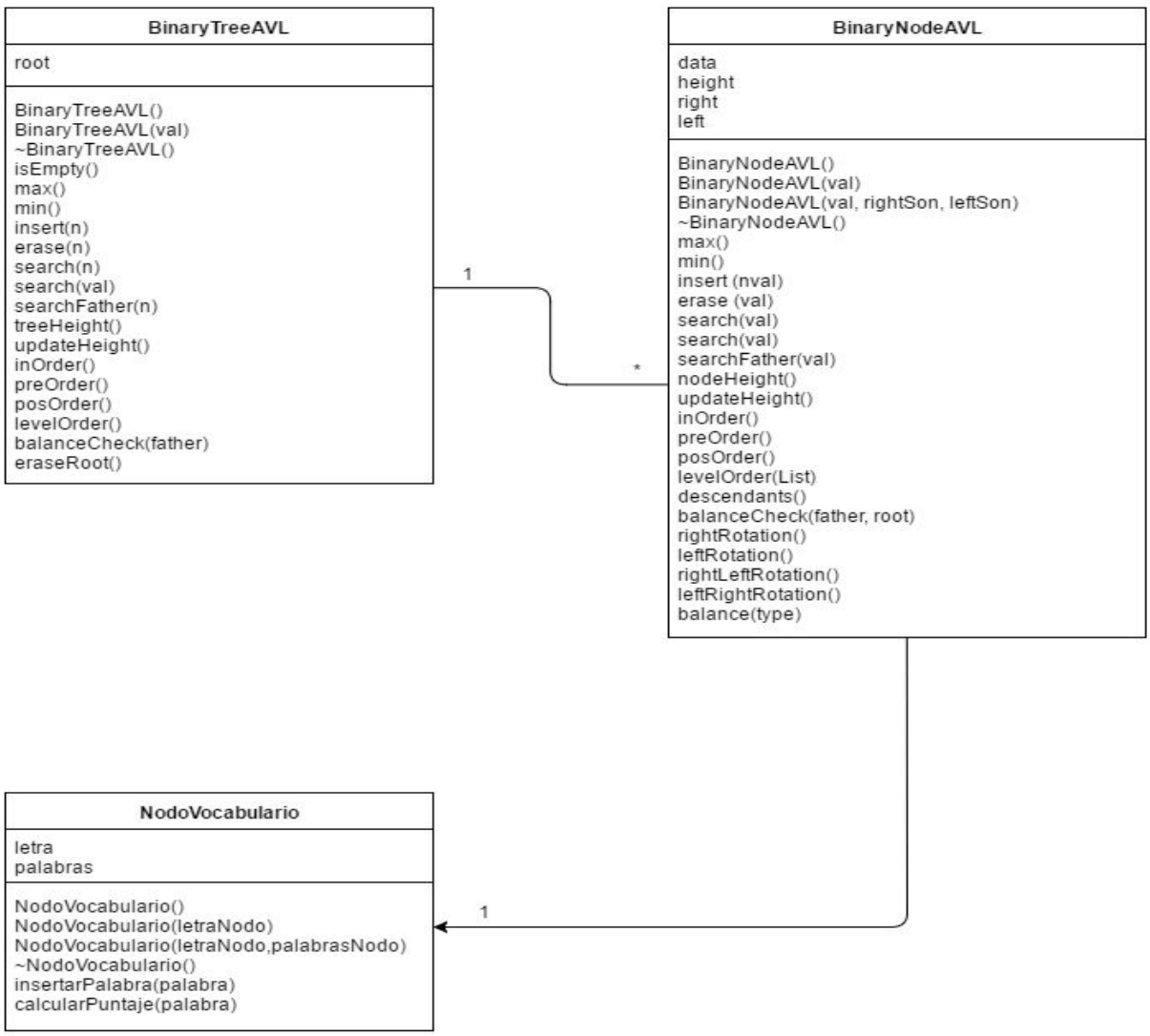
- root, apuntador de la raiz del arbol.

Operaciones:

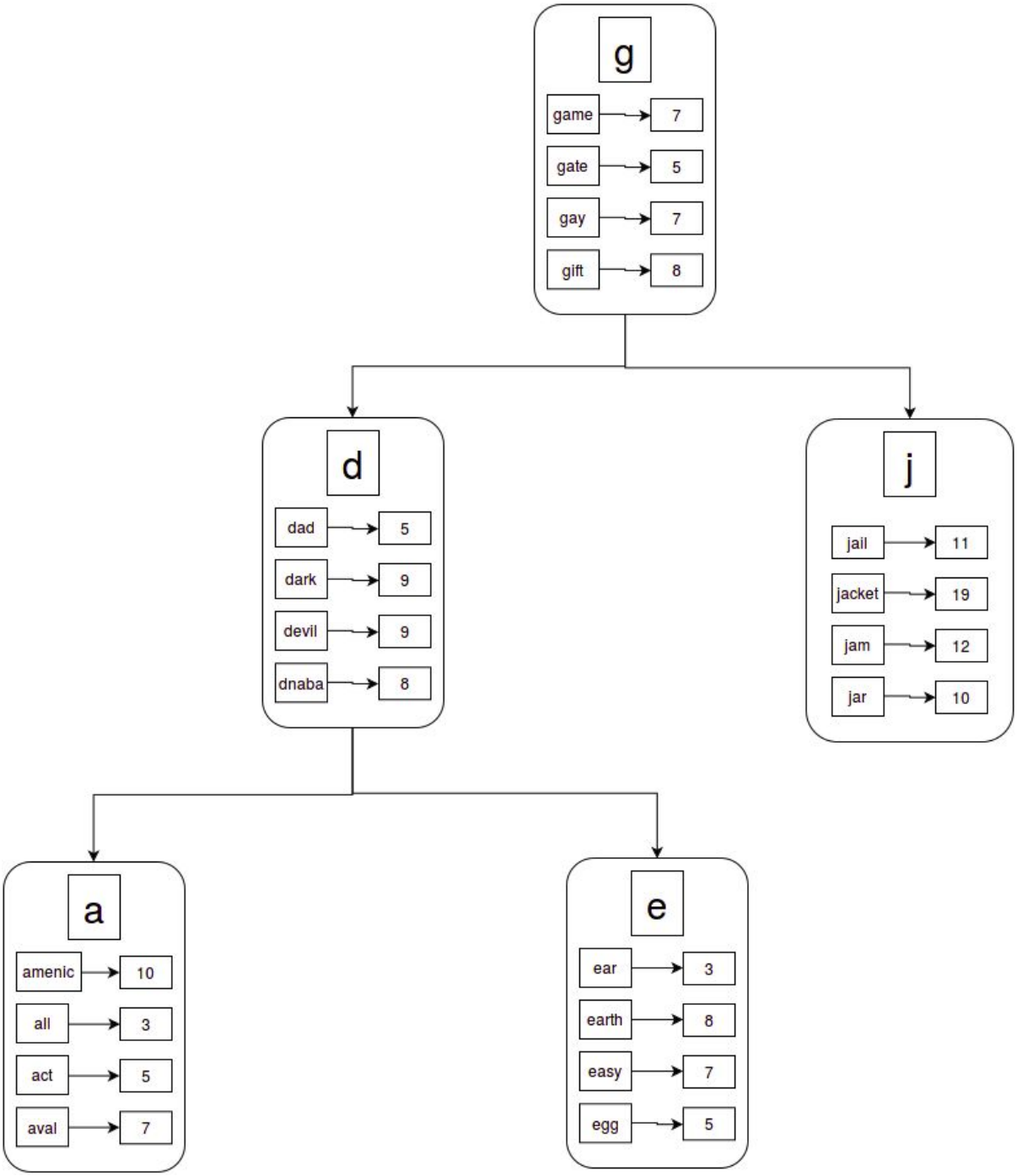
- BinaryTreeAVL(), constructor del árbol vacío.
- BinaryTreeAVL(val), constructor del árbol con raíz igual a val.
- ~BinaryTreeAVL(), destructor.
- isEmpty(), verifica si el árbol está vacío.
- getRoot(), retorna un apuntador a la raíz.
- setRoot(nroot), fija el valor de la raíz a nroot.
- max(), retorna un apuntador al nodo con mayor valor en el árbol.
- min(), retorna un apuntador al nodo con el menor valor en árbol.
- insert(n), inserta un nodo al árbol, siempre y cuando este no esté ya dentro del árbol.
- erase(n), elimina el nodo con el mismo valor que n, si lo encuentra.

- search(n), retorna verdadero si encuentra el nodo dentro del árbol, sino retorna falso.
- search(val), retorna un apuntador al nodo que contenga un carácter igual a val.
- searchFather(n), busca si el nodo con el mismo valor de n tiene padre.
- treeHeight(), retorna la altura del árbol.
- updateHeight(), recalcula la altura del árbol.
- inOrder(), recorrido en inOrden del árbol.
- preOrder(), recorrido en preOrden del árbol.
- posOrder(), recorrido posOrden del árbol.
- levelOrder(), recorrido por niveles del árbol.
- balanceCheck(father), balancea el árbol.
- eraseRoot(), elimina la raíz del árbol.

Diagrama de relación entre TAD:



Representación gráfica de la organización de los datos:



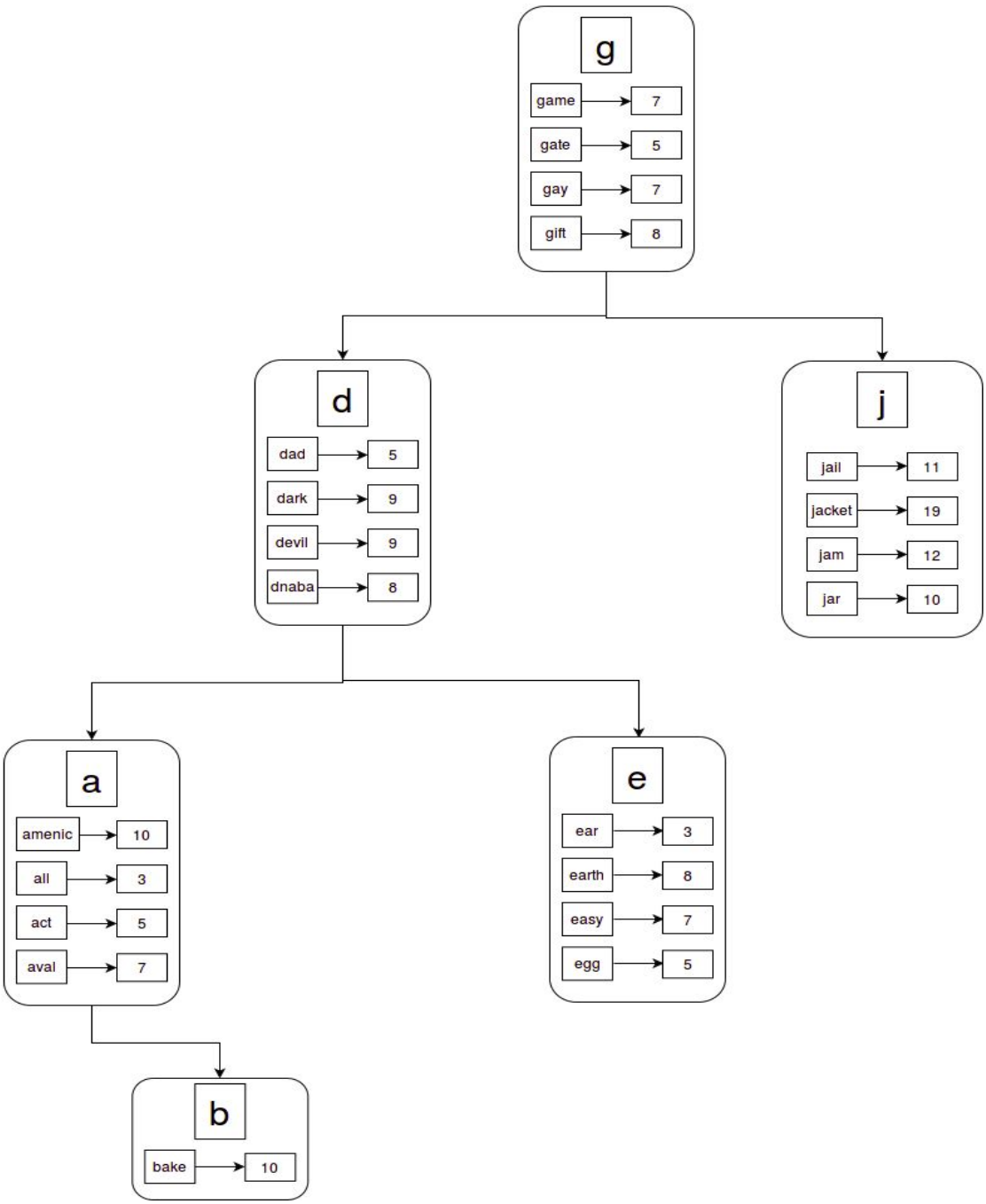
- Esta sería una representación gráfica del diseño que se aplicó sobre algunos datos que podrían estar en un tiempo dado dentro de los datos a guardar. Como se puede observar, dentro de cada nodo existen dos elementos: la letra que indica por cual letra empiezan las palabras y un conjunto de palabras con su respectivo puntaje asociado.

- En los casos de la palabras invertidas, estas también se encuentran en el nodo que indique la primera letra al invertir la palabra. Un ejemplo de ello son las palabras *lava, cinema* y *aband.* Tomando el caso de la palabra *lava,* al realizar la operación de invertir la palabra quedaría como *aval,* ya que esta es la palabra que se guardara en el diccionario corresponde al nodo de la letra *a.*

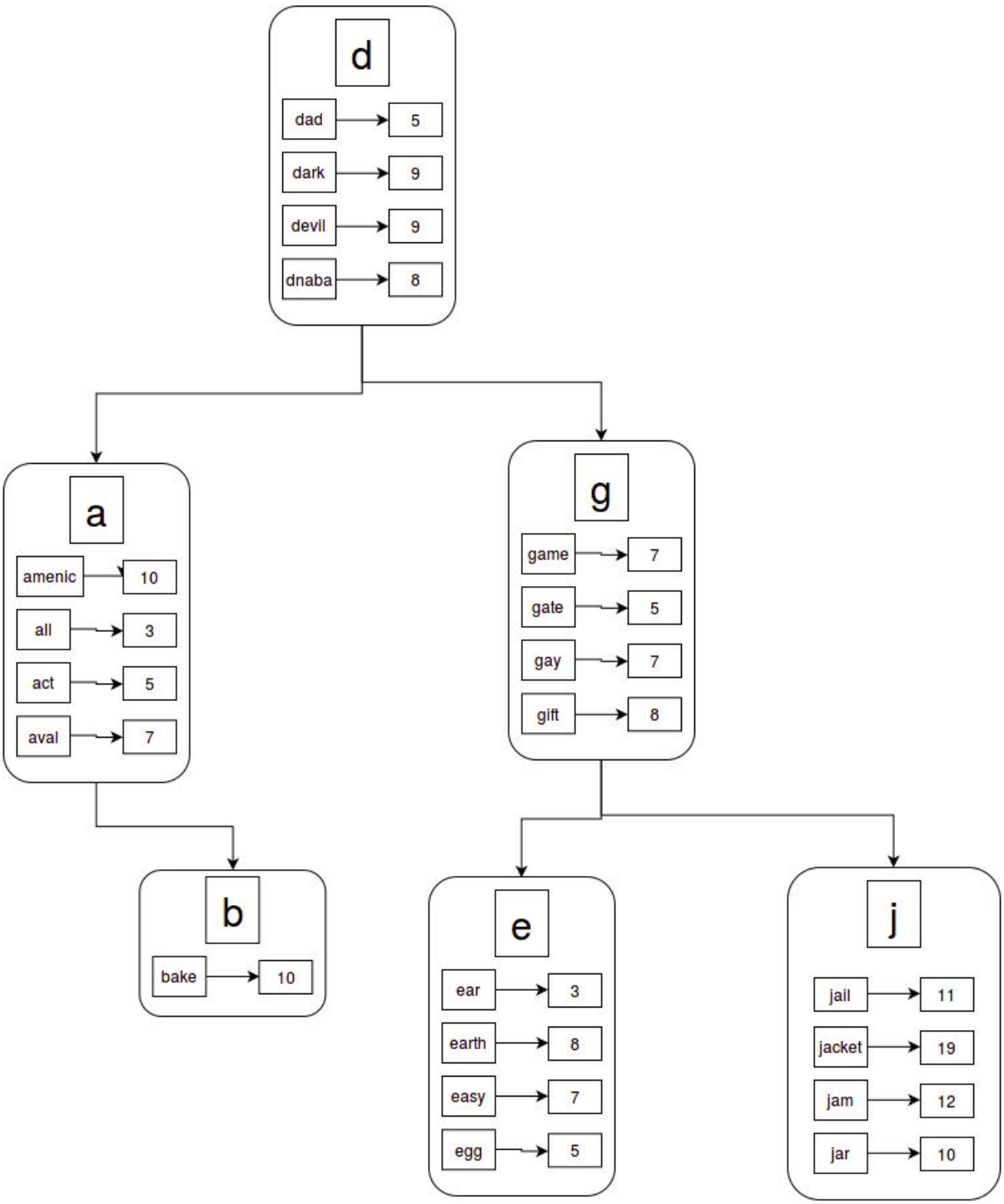
➔ Tomando en cuenta el gráfico anterior, vamos a realizar las operaciones básicas que conciernen al componente: insercion de una nueva palabra y búsqueda del valor de la palabra dada.

◆ **Inserción de una palabra:**

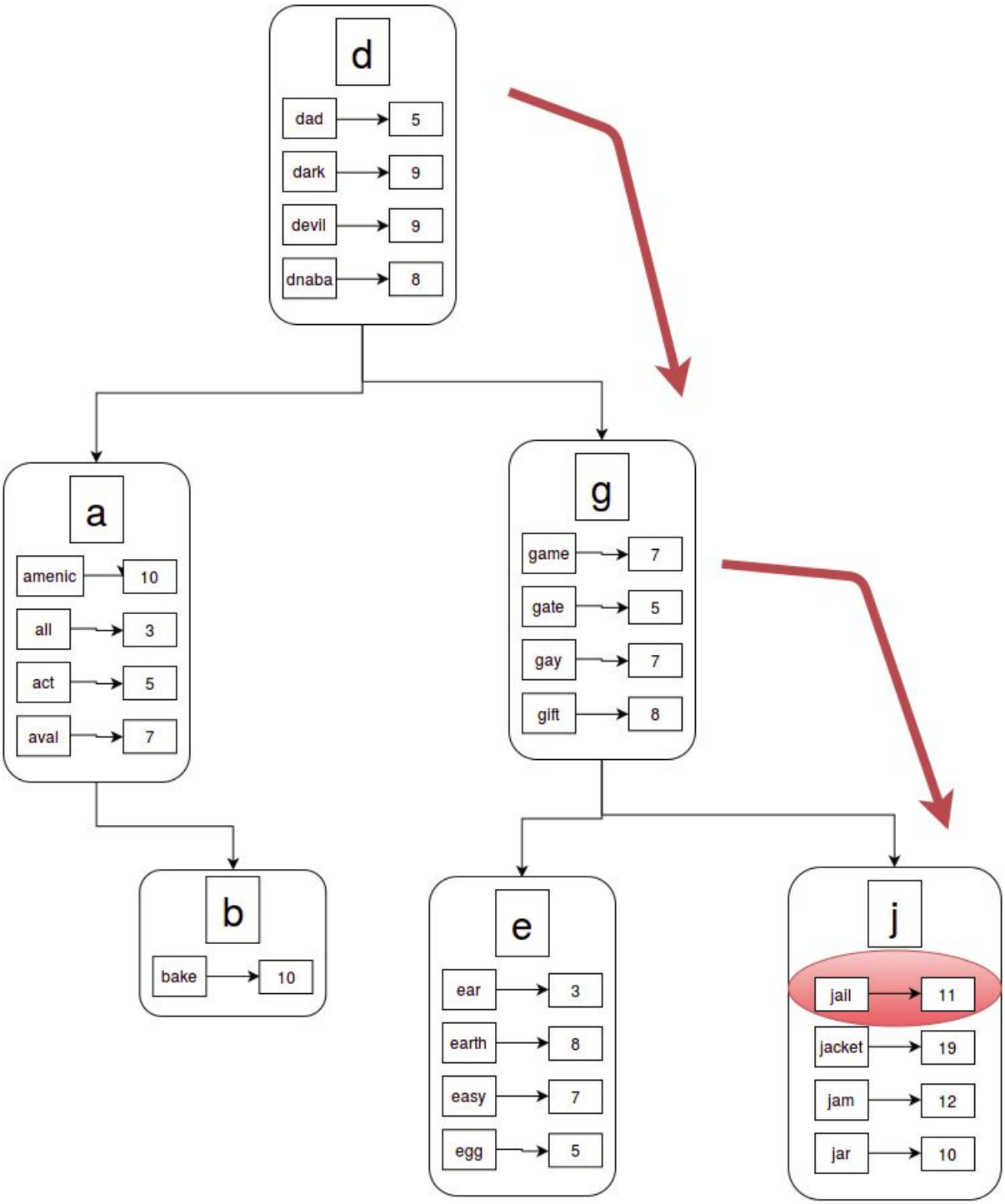
- Para este caso vamos a insertar la palabra *bake.* Ya que no se encuentra el nodo de la letra *b* dentro del árbol se crea y se adiciona al árbol. Posteriormente se inserta la palabra con su valor asociado al nodo creado. Para este ejemplo el árbol luego de la inserción quedaría de la siguiente forma:



- Como se puede observar se ha insertado el nuevo nodo con su palabra y el valor asociado a esa palabra. Sin embargo, dado que se utilizó un árbol AVL en donde se realizan balanceos sobre el árbol cada vez que se insertan datos y se sobrepasa cierta diferencia entre las alturas de los hijos, la realidad es que el árbol en definitiva queda como se muestra a continuación:



- **Búsqueda del valor de una palabra dada:**
  - Para caso vamos a buscar la palabra *jam* para obtener su puntaje respectivo. Primeramente se realiza la búsqueda del nodo que contiene la palabra, es decir, el nodo cuya letra sea igual a la primera letra de la palabra a buscar. Una vez encontrado dicho nodo, se realiza la extracción del valor asociado a la palabra buscada.



➔ **Aclaraciones sobre el diseño:**

- Las palabras almacenadas dentro de cada nodo se encuentran guardadas en letras minúsculas al igual que la letra del nodo. Esto quiere decir que al insertar una palabra o un nodo nuevo se convierte a minúsculas para poder tener un estándar dentro del diseño.
- Palabras con caracteres inválidos no son insertados en el árbol. Al encontrar una palabra con estas condiciones se escribe por pantalla el error respectivo.
- Al ser la estructura de un árbol AVL, se continúan aplicando las rotaciones respectivas para mantener el balanceo del mismo. Dentro de cada nodo existe una característica de altura que guarda la altura del nodo; esto para facilitar el desarrollo de las rotaciones. A pesar de que no aparecen en los diagramas presentados dicha característica, existen dentro de cada nodo del árbol pero no son relevantes para la solución del componente.
- Las palabras dentro de cada nodo no están repetidas ya que la programación y la estructura utilizada no permite palabras repetidas dentro de la estructura. Si se encuentra una palabra que cumpla con esta condición se realizará el mensaje de error correspondiente.