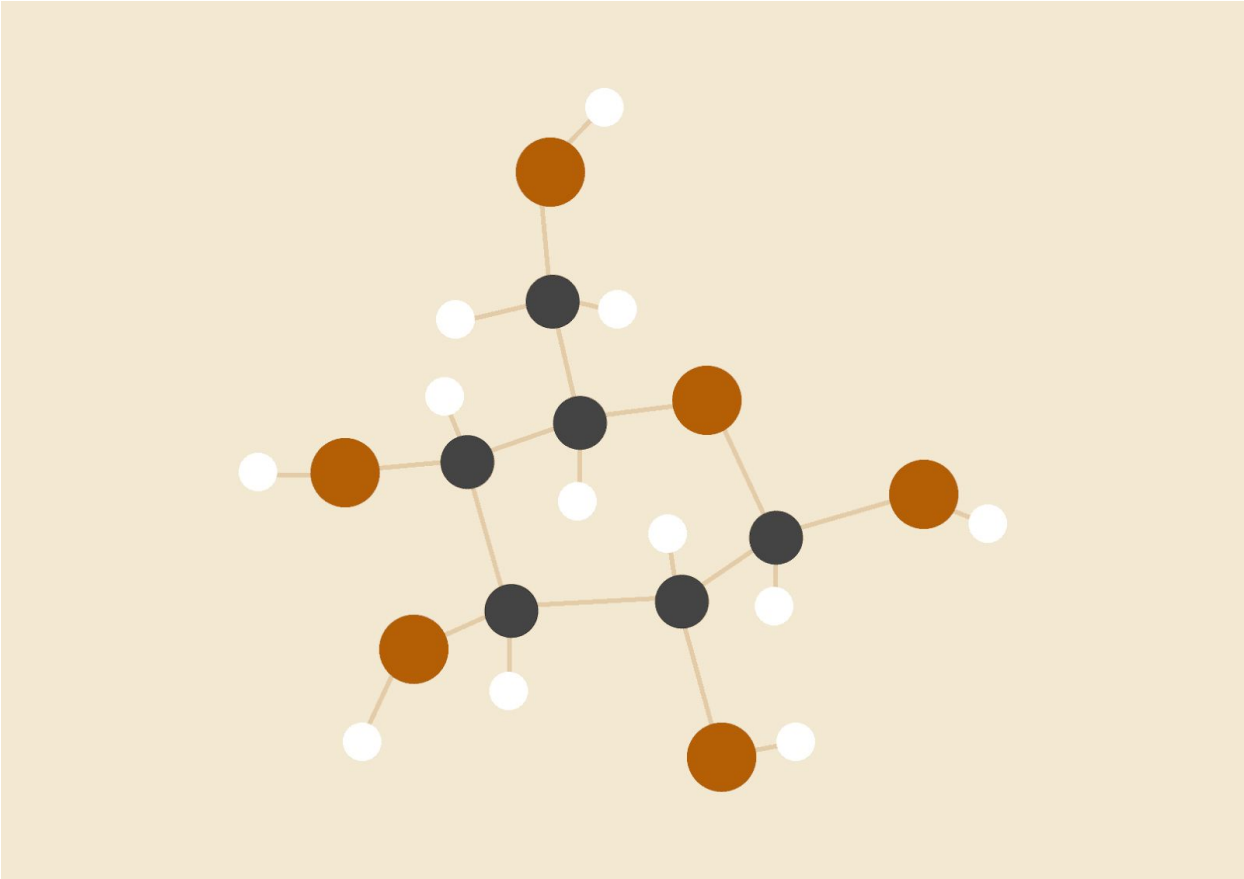
LABORATORIO Nº3: ÁRBOLES



RODRIGO MARDONES AGUILAR

ANÁLISIS DE ALGORITMOS

Y

ESTRUCTURA DE DATOS

ÍNDICE

INTRODUCCIÓN

En las ciencias de la computación, las estructuras de datos son, como su nombre lo indica, estructuras o formas de ordenar y organizar datos para que estos puedan representar de mejor manera la información que necesitamos obtener de algún problema específico. Estas estructuras simplifican de sobremanera la resolución de aquellos problemas que, con tipos de datos simples no podemos modelar.

Los árboles son estructuras que nos permiten modelar problemas de distinta índole, apoyado en la teoría de grafos, estas son estructuras que, como su nombre lo indica, poseen forma de árbol, al poseer una raíz, y hojas en sus extremos, las que al mismo tiempo, también pueden comportarse como árbol.

El objetivo general de este informe es resolver el problema planteado en el enunciado nº3 de laboratorio, el cual consiste en el modelado de un diccionario inglés-español, donde se pueda buscar palabras y buscar su

Para esto los objetivos específicos a completar son los siguientes:

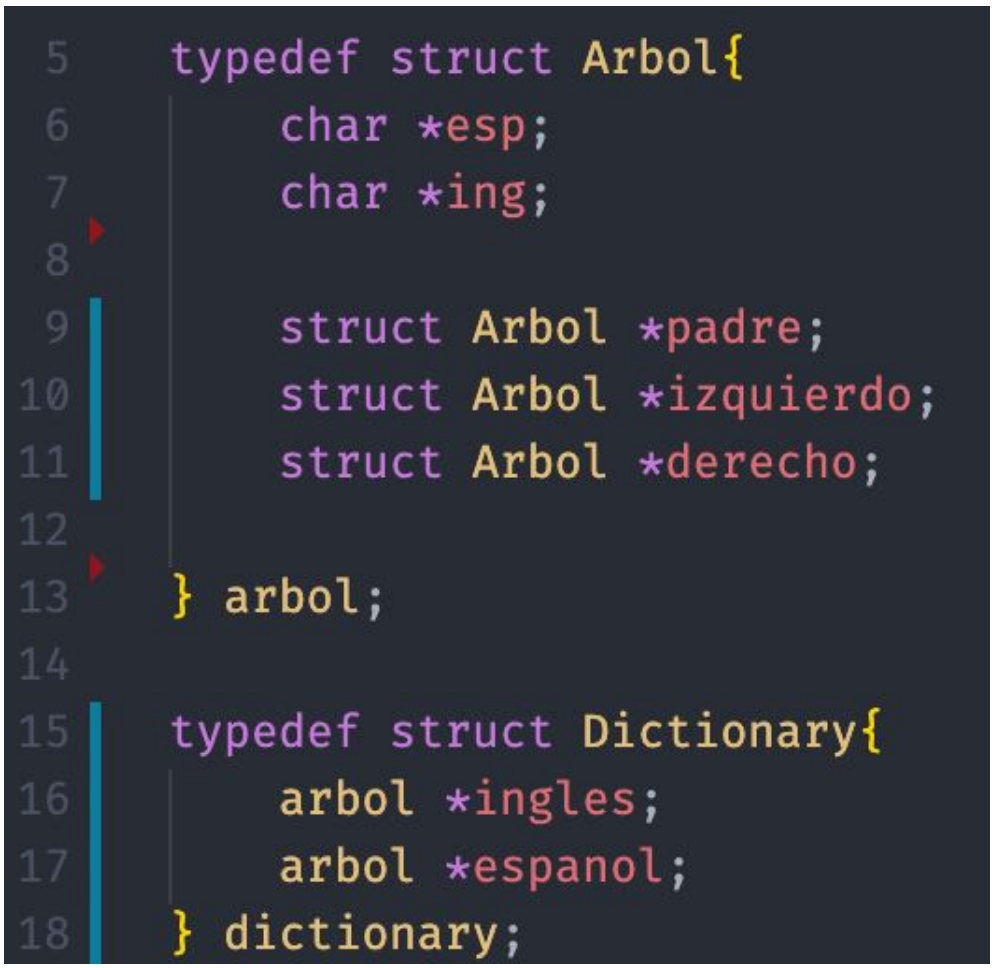
- crear TDA de datos necesarias para modelar un diccionario lenguaje-español
- definir funciones asociadas a cada TDA para su utilización en la solución
- plantear solución con algún algoritmos de búsqueda
- ocupar algoritmo y ver si satisface las necesidades del problema
- análisis de resultados

DESCRIPCIÓN DE LA SOLUCIÓN

Definición de TDA

Para el problema en cuestión, se plantean las siguientes estructuras de datos que nos ayudarán a modelar el problema como un grafo:

- árbol: estructura de base del árbol, donde se contiene los elementos del árbol correspondiente, palabra en inglés y español, árbol padre, hijo izquierdo e hijo derecho.
- dictionary: estructura que contiene los dos árboles para contener los árboles, tanto “inglés-español” como “español-inglés”.



definición de funciones de TDA

Para los TDA anteriores se han creado distintas funciones que cumplen las necesidades de constructor, accesor, modificador y otros. Estos son:

- crearArbol: función que nos permite crear un árbol con sus debidos parámetros.
- agregarHoja: función que nos permite agregar una hoja al árbol vacío o ya con hojas existentes.
- crearDiccionario: función que nos permite crear un los dos árboles para el caso de estudio y los completa con la entrada del archivo correspondiente
- buscarPalabra: función que permite buscar una palabra dentro del árbol correspondiente si existe y nos retorna la hoja con toda su información.
- buscarPadre: función que nos permite buscar el padre de una palabra si existe dentro del árbol y nos retorna su hoja con todos sus datos correspondientes.

Cabe destacar que todas las entradas y salidas de cada función de definición para los TDA se encuentran debidamente documentadas en el archivo de cabecera correspondiente.

Descripción del problema y solución

Desde el punto de vista del usuario, lo que se necesita es un simple diccionario de datos en español-inglés y viceversa donde cada palabra se contiene de manera inversa, además de contener ambos diccionarios ordenados alfabéticamente, para una mayor compresión y búsqueda.

Desde el punto de vista técnico, indicaremos que cada palabra en nuestro diccionario será considerada como una “hoja” de un árbol, lo que a su vez contendrá información relacionada a sí misma y a las “hojas” que lo continúan. Al ser un árbol ordenado, siempre contendremos ordenado alfabéticamente el árbol al ir comparando cada palabra e ir agregando según el valor de de sus caracteres en ASCII.

El problema en cuestión consiste en la búsqueda en orden de cada palabra, su significado en español y viseversa, es lo que se conoce como “búsqueda binaria”, esta consiste en la búsqueda de un elemento en específico desde el centro hacia los extremos, para poder reiterar así mismo nuevamente con las mitades de cada una de forma recursiva hasta dar con el elemento específico.

Para el caso siguiente, se ha propuesto el siguiente algoritmo:

```
arbol *buscarPalabra(arbol *a, char *palabra, int flag){
    if(a == NULL){
        return NULL;
    }
    if(flag){
        if(strcmp(a->esp, palabra) == 0){
            return a;
        }
        if(strcmp(a->esp, palabra) < 0){
            return buscarPalabra(a->izquierdo, palabra, flag);
        }
        if(strcmp(a->esp, palabra) > 0){
            return buscarPalabra(a->derecho, palabra, flag);
        }
    } else {
        if(strcmp(a->ing, palabra) == 0){
            return a;
        }
        if(strcmp(a->ing, palabra) < 0){
            return buscarPalabra(a->izquierdo, palabra, flag);
        }
        if(strcmp(a->ing, palabra) > 0){
            return buscarPalabra(a->derecho, palabra, flag);
        }
    }
}
```

La búsqueda se hace de manera recursiva, viendo el orden de la palabra y comparándola con el nodo visitado y viendo si es necesario visitar su izquierda o derecha respectivamente. Para el caso este es una versión modificada para comparar por palabra y para indicar el tipo de búsqueda, si se debe buscar la palabra en español o inglés con “flag”.

ANÁLISIS DE LOS RESULTADOS

Una vez completado lo anterior podemos indicar que nuestro TDA cumple con lo pedido, la búsqueda de palabras en un diccionario en inglés-español y español-inglés, indicando padres, hijos para cada uno de los idiomas.

Análisis de complejidad algorítmica

Para este caso haremos el análisis se hará en la búsqueda binaria contenida en el código, la cual se ubica en la búsqueda de la palabra correspondiente en el algoritmo puesto anteriormente, entonces para el caso de la complejidad algorítmica al ser un algoritmo recursivo tenemos los siguientes casos

Para casos donde se encuentra el nodo:

$$T(n) = 1$$

Para casos donde se debe hacer un recorrido tanto a izquierda o derecha:

$$T(n) = T(n/2) + 1$$

Para casos donde se debe hacer más de un recorrido recorrido tanto izquierda como derecha:

$$T(n) = (T(n/2) + 1) + (T(n/4) + 1) + ... + (T(n/2^{k}) + 1$$

Donde n es el numero de hojas de un arbol. por lo tanto el valor de “k” tiende a $Log(n)$. Es por esto que podemos concluir que para el caso anterior se tiene que:

$$O = Log(n)$$

Consideraciones

se decidió mantener esta representación del diccionario para mantener un orden, pues si bien se guardan efectivamente ambos diccionarios de manera correcta y ordenada, no cumple con mantenerlo en un solo nodo para el árbol sino que se generan dos árboles totalmente separados para cada lenguaje.

CONCLUSIONES

Podemos finalizar este informe concretando que, en base al análisis presentado de los resultados, concretamos todos los objetivos específicos planteados al comienzo de todo.

Hemos podido modelar el problema de modo que la solución pueda ser encontrada, ahora bien, no es la más óptima por lo que es necesario, se podría juntar ambos árboles para poder dar solución definitiva al tema de espacio y de recorrido ordenado.

La búsqueda binaria en este caso es la más eficiente para dar solución a este tipo de problemas, puesto que plantea una búsqueda dicotómica, basada en el valor contenido y en el tamaño del árbol.