TEC | Tecnológico de Costa Rica

# Proyecto#1

Fabrizio Alvarado Barquero.
2017073935.

Emanuelle Jimenez Sancho.
2017136727.

Instituto Tecnológico de Costa Rica

Lenguajes de programación.

Maria Auxiliadora Mora.

18/09/19

## Tabla de contenidos:

## Descripción del problema:

- Primera parte:

La primera parte del proyecto es hacer una estructura de datos del árbol. El árbol debe de tener las siguientes funcionalidades: Insertar, eliminar, buscar, encontrar el ancestro, buscar el hermano derecho, buscar el hermano izquierdo, imprimir todos los nodos, imprimir todos los nodos de una rama.

- Segunda parte:

En la segunda parte del proyecto tenemos un problema a cerca de un juego muy simple, el dominó, principalmente había que encontrar la cadena de dominós más larga a partir de una lista de fichas dadas. También habían dos pequeños problemitas existentes, los cuales eran el añadir o eliminar fichas de una lista para que el usuario pueda jugar con las fichas que le plazca.

## Diseño del programa:

- Primera parte:

Para el diseño del árbol se implementó un struct de la siguiente manera para los nodos:

( id del nodo - id del padre - lista de hijos - datos del nodo )

Va existir una variable global que se llamará “arbol”, esta será una lista con todos los nodos del árbol, cada vez que se hace un insert o delete, se altera el contenido del mismo.

Optamos por esta solución porque es muy fácil implementar un árbol en forma de lista. Como bien se ve en clases todo en el lenguaje funcional se maneja por listas, es por eso que nos vimos influenciados por esa ideología y decidimos implementarlas de tal forma, que al final resultó bastante provechosa.

**Algoritmos utilizados**

En esta sección se van a explicar todos los algoritmos que utilizamos en cada una de las funcionalidades del programa.

- List-all-nodes: Para esta función nosotros llamamos a la función de list-some-nodes, pero con el id del nodo raíz como parámetro, en este caso reutilizamos código.
- List-some-nodes: Esta función recibe como parámetros el id del nodo raíz y el árbol en general. Nosotros llamamos a una sub-función que se encarga de buscar el nodo con ese id en específico e imprimirlo y de igual forma se llama a otra función que recibe los hijos de ese nodo. Dicha subfunción recorre la lista con los hijos y vuelve a llamar a List-some-nodes con el id de cada uno de los hijos.
- Find-node: Esta función fue muy sencilla de programar, simplemente recorre la lista (árbol) y si el primero de la lista, que es un nodo, tienen los mismo ids, entonces lo retorna.
- Insert-node: Esta función recibe como parámetro los datos del nodo y el árbol. La misma hace un append del nodo a la variable global y luego llama a una subfunción que se encarga de recorrer la lista y agregar el id del nodo insertado al padre (si no se encuentra el programa retorna paréntesis vacíos haciendo referencia que no hay datos en el mismo)

- Delete-node: Esta fue una de las funciones más complicadas durante nuestro transcurso del proyecto. Al principio la función recorre el árbol y cuando encuentra el nodo entonces llama a la función de “delete-first-node”, que como el nombre lo dice, elimina el primer nodo del árbol recorrido. Una vez dentro de la función de “delete-first-node”, se llaman a dos subfunciones más, “delete-node-loop” que recibe los hijos del nodo y el árbol, esto es necesario porque primero se deben de eliminar los hijos y luego el nodo actual. Después de eliminar los hijos la función llama a “delete-first-from-tree”, que simplemente elimina el nodo actual de la variable global.
- Ancestor: Esta función fue la más fácil de implementar, simplemente se recorre el árbol (la lista de nodos) y si el id es el mismo al id del nodo actual, entonces haga un “find-node” con el padre de ese nodo.
- Find-right-sibling y Find-left-sibling: Estas dos funciones son muy parecidas. Ambas llaman a una subfunción que recibe como parámetros lo siguiente: Lista de hijos del padre del nodo a buscar del hermano y el id del nodo. Después de tener la lista de los hijos del padre, se realizan varias verificaciones. Si la lista es de tamaño 1, es porque no tiene hermanos derechos o izquierdos. Si la lista es de tamaño 2 entonces tiene al hermano a la par. Sino entonces llame a la misma función, pero sin el primer elemento de la lista. Cabe rescatar que buscar el hijo izquierdo hace un recorrido al revés (De atrás hacia adelante).

- Segunda parte:

Para el diseño del juego de dominó se optó por crear inicialmente las 28 fichas por defecto (que reciben los nombres de: f##, con un formato utilizado así: (list # #)), tal como lo indicaba el documento de especificación del proyecto, seguidamente se crearon 2 listas por defecto de igual forma (lista-por-defecto y lista-alternativa), donde una contiene las 28 fichas previamente creadas y la otra está vacía, estas dos se crearon con el fin de probar la funcionalidad de añadir o eliminar fichas.

Se decidió utilizar la estructura de lista tanto para las listas de fichas, como para la fichas mismas ya que creemos que es la estructura más versátil y simple de manipular en el lenguaje.

**Algoritmos utilizados**

El juego en si únicamente tiene dos funciones, las cuales son:

- jugar | lista cadena iteraciones
  Esta función como su nombre muy bien lo indica es la función inicial del juego y tiene 2 propósitos , el primero es verificar si quedan jugadas restantes mediante la variable “iteraciones”, que una vez que esta variable sea igual que el largo de la lista de fichas, el juego termina y retorna la cadena más larga encontrada.

- verificar-extremos | lista cadena iteraciones
  Esta se puede decir que es el núcleo del programa, ya que aquí es donde se validan las 4 posibles jugadas que se pueden hacer con la primera ficha de la lista, de ser el caso que no encuentre ninguna jugada válida, únicamente agrega la ficha al final de la lista misma y llama recursivamente a la función de jugar, la cual se explicó arriba detenidamente.

## Análisis de resultados:

- Primera parte:

  *Objetivos que se alcanzaron:*

  - Todas las funciones del árbol sirven correctamente. Se pueden hacer operaciones básicas como insertar, eliminar y buscar. De igual forma como las extra que fueron solicitadas en los requerimientos, las cuales son: Buscar hermano derecho de un nodo en específico, Buscar hermano derecho de un nodo en específico, imprimir todo el arbol, imprimir el árbol con una hoja de parámetro.

  *Objetivos que no se alcanzaron:*

  - Proyecto exitoso, ningún objetivo quedó incompleto.

- Segunda parte:

  *Objetivos alcanzados:*

  - Existe un mecanismo de lectura de las fichas de entrada.
    - Se pueden incluir fichas.
    - Se pueden eliminar fichas.
  - El sistema retorna la lista más larga de fichas que pueden conectar según las reglas de dominó.

  *Objetivos que no se alcanzaron:*

  - Proyecto exitoso, ningún objetivo quedó incompleto.

## Manual de usuario:

- Primera parte:

En primera instancia se explicara como funciona el nodo raiz. Este tendra como nodo padre el 0, para que el programa sirva se tiene que insertar un nodo con este padre. Despues se va a asumir que todos los ids de los nodos son unicos, esto para evitar problemas con los algoritmos. Un ejemplo de insertar serian de la siguiente manera:

(insert-node 1 0 “datos del nodo” arbol)
(insert-node 2 1 “datos del nodo” arbol)
(insert-node 3 1 “datos del nodo” arbol)
(insert-node 4 2 “datos del nodo” arbol)
(insert-node 5 2 “datos del nodo” arbol)
(insert-node 6 2 “datos del nodo” arbol)
(insert-node 7 3 “datos del nodo” arbol)

Cada nodo tiene un parametros de datos, que sirven para almacenar datos en el mismo, de igual forma recibe como parametro el arbol, que viene siendo la lista de nodos. Como se puede observar, el nodo raiz es el 1, porque tiene como id del padre el 0.

A continuacion se explicaran las demas funciones del programa:

- List-all-nodes: Esta funcion tiene la siguiente sintaxis

  (list-all-nodes arbol)

  Recibe como parametros el arbol y los imprime.

- List-some-nodes: Esta funcion tiene la siguiente sintaxis

  (listsome-nodes 1 arbol)

  Recibe como parametros el id del nodo a imprimir (que en este caso es el nodo raiz) y tambien el arbol.

- Find-node: Esta funcion tiene la siguiente sintaxis:

    (find-node 4 arbol)

    Recibe como parametro el id del nodo que se desea buscar y tambien el arbol.

- Delete-node: Esta funcion tiene la siguiente sintaxis:

    (delete-node 1 arbol)

    En este caso la funcion recibe el nodo a eliminar, que es el nodo raiz y de igual forma recibe el arbol.

- Ancestor: Esta funcion tiene la siguientes sintaxis:

    (ancestor 4 arbol)

    Recibe el nodo a buscar, en este caso deberia devolver el nodo 2, segun el arbol que creamos anteriormente y tambien el arbol. No se puede buscar el ancestor de 1, porque no tiene padre.

- Find-right-sibling y Find-left-sibling: Estas funciones depende mucho en el orden que inserto los nodos, siempre los inserta a la derecha de ellos, por ejemplo: El 3 esta a la derecha del 2 (en el caso de arriba). Tiene la siguiente sintaxis:

    (find-right-sibling 2 arbol)

    Esta funcion deberia retornar 3, si no tiene hermano entonces devuelve “no hay hermano”.

- Segunda parte:

Para iniciar la aplicación pero se debe de elegir una lista de ficha, ya sea la que está por defecto (lista-por-defecto), con las 28 fichas previamente creadas, a la cual también se le pueden eliminar fichas (eliminar-ficha), o una lista alternativa (lista-alternativa) la cual esta vacía y se le deben agregar las fichas deseadas(anadir-ficha).
Una vez esté listo el conjunto de fichas que se va a utilizar para jugar, se debe de llamar a la función jugar, seguidamente de los parámetros “lista” (la cual es la lista que

se definió anteriormente para jugar), cadena que es únicamente una lista vacía ((list)) y finalmente el parámetro “iteraciones”, que siempre va a ser 0 al iniciar el juego.
Una vez se corra la función va a retornar la cadena más larga de fichas en forma de lista ordenada.

## Conclusion personal:

- Conclusión de Emanuelle:

A mi parecer este proyecto me gusto bastante por el hecho que nos puso a pensar en cómo se implementa una estructura de datos, que en este caso es el árbol, en un paradigma de programación funcional. Anteriormente lo habíamos hecho en imperativo y me llamó mucho la atención implementarlo en un ambiente nuevo. Cabe rescatar que el proyecto no fue de mucho desafío, no fue un reto para mí poder hacer un árbol o implementar el algoritmo del domino, es por eso que una dificultad mayor sería interesante de programar.

- Conclusión de Fabrizio:

Creo personalmente que el proyecto me ayudó mucho a poner los pies en la tierra como llaman vulgarmente, ya que al inicio no sentía que en verdad conocía el lenguaje y a terminar de entender bien el paradigma, más allá de la teoría.
Creo que también me ayudó mucho a reforzar los conceptos y la práctica de la recursión, ya que últimamente estaba muy acostumbrado a utilizar iteración.
Finalmente creo que fue un reto difícil mas no inalcanzable.