Minutas De Proyecto

Comisión n°: 33
Integrantes

- Aguirre, Luciano(LU: 99189).
- Alimenti Bel, Traian(LU: 99421).

Limitaciones del programa

El programa ha sido testeado de manera rigurosa y funciona correctamente. A pesar de poseer una interface gráfica sencilla, posee una gran robustez, esto se debe a que siempre se controla que la información ingresada por el usuario sea del tipo correcto, es decir, sean números enteros. En el caso que el usuario ingrese por equivocación cualquier otro tipo de dato (String, Float, etc.), el programa muestra carteles de advertencia para avisar del error ocurrido.

Hemos asumido a la hora de realizar nuestro programa, que en el árbol no habrá nodos con rótulos repetidos, si se ingresan dos o más nodos con el mismo rótulo, es posible que ocurran fallas en algunos métodos (por ejemplo en el mapeo), es decir, ejecutan pero no realizan su tarea de manera correcta.

Todos los métodos se hacen en base a un árbol no vacío, por esto es que hasta que el usuario no carga un árbol en el programa, el resto de los botones quedan inhabilitados.

Problemas en el desarrollo del proyecto

Los únicos problemas que tuvimos a la hora de desarrollar el proyecto estuvieron relacionados con la interface grafica. Nuestra idea era crear una barra de menús para poder agrupar de alguna manera los botones en menús, y de que esta manera todo quede más organizado, pero a la hora de ejecutar el programa, no funcionaban correctamente los oyentes al hacer click, por lo que los métodos no siempre ejecutaban al clickear sobre ellos. Por esta razón, decidimos no hacer una interface muy avanzada, y dejar una sencilla que sea fácil de utilizar para el usuario y que realice todas las tareas a la perfección.

Soluciones ejercicios 6-13

A continuación se encuentran las soluciones correspondientes a los ejercicios desde el inciso 6 hasta el inciso 13. Las soluciones se plantean de manera informal, en forma de algoritmo, es decir, se indican sencillamente los pasos necesarios para poder resolver cada ejercicio.

Ejercicio 6: Mostrar los rótulos de los nodos en orden descendente.

Nuestra idea a la hora de solucionar este problema fue la siguiente:

- Pedir el iterador del árbol para tener todos los rótulos de los nodos.
- Crear una cola con prioridades auxiliar con un comparador de enteros.
- Por cada rótulo en el iterador, ingresar un par (rótulo,null) a la cola con prioridades.
- Crear una pila auxiliar.*
- Mientras la cola con prioridades no este vacía, insertar en la pila la clave de la entrada obtenida al ejecutar removeMin().
- Crear una lista.
- Mientras la pila no este vacía, desapilar y agregar al final de la lista el rótulo desapilado.

De esta manera al finalizar, tenemos en la lista los rótulos ordenados en forma descendente, la devolvemos y luego se muestra su contenido al usuario.

*Se utiliza una pila ya que sirve para invertir secuencias o colecciones. Al pedir a la cola con prioridades el removeMin(), nos devuelve el rótulo mas pequeño, entonces al insertarlo en una pila al finalizar los rótulos quedan en el orden deseado.

Ejercicio 7: Eliminar todos los nodos de un nivel dado. El nivel se recibe como parámetro. Retornar los rótulos de los nodos eliminados en una pila y mostrar su contenido.

Para resolver este problema tuvimos en cuenta una condición inicial: si el árbol no posee el nivel que es recibido por parámetro, se indica esta situación al usuario a través de un cartel y no se ejecuta el método.

Si dicho nivel se encuentra en el árbol procedimos a realizar lo siguiente:

- De manera similar a lo que se realiza para mostrar un árbol por niveles, lo que hicimos fue pedir un iterador de

- posiciones del árbol, crear una cola auxiliar y encolar inicialmente la raíz del árbol y un null. Además llevamos un contador para saber en que nivel nos encontramos y frenar cuando lleguemos al deseado.
- Mientras la cola auxiliar no este vacía desencolamos y se lo asignamos a una variable. Si el valor de la variable es distinto de null, es decir, es un nodo, encolamos todos los hijos de este nuevamente en la cola. Si es null, pasamos a otro nivel, por lo que aumentamos el contador en uno, y si la cola no esta vacía encolamos otro null.
- Al salir de este ciclo, tenemos en la cola auxiliar todas las posiciones de los nodos que se encuentran en el nivel que se desea eliminar. Creamos una pila auxiliar para almacenar los rótulos de los nodos eliminados. Entonces mientras la cola no este vacía y no desencolemos un null, desencolamos, insertamos el rótulo del nodo en la pila y llamamos al método removeNode(…) de árbol para eliminar cada nodo.

Al finalizar, hemos eliminado del árbol el nivel ingresado por el usuario, retornamos la pila y se muestra el contenido almacenado en la misma al usuario.

Ejercicio 8: Dados dos rótulos R1 y R2, retornar y mostrar el camino del nodo N1 que tiene como rótulo a R1 al nodo N2 que tiene como rótulo a R2.

Lo primero que hacemos es encontrar los nodos cuyos rótulos son los R1 y R2. Si alguno de los dos no pertenece al árbol, se informa al usuario a través de un cartel y el método no se ejecuta. Si ambos pertenecen al árbol, los asignamos dos variables N1 y N2 y resolvemos el resto del problema de la siguiente manera:

- Creamos dos pilas auxiliares pila1 y pila2. En la pila1, mientras N1 sea distinto de la raíz, apilamos N1 y a este le asignamos su padre. Con la pila2 realizamos lo mismo, mientras N2 sea distinto de la raíz, apilamos N2 y a este le asignamos su padre. Al finalizar tenemos en cada una de las pilas los ancestros de cada nodo, excepto la raíz. Esta la asignamos a una variable especial donde almacenamos el último nodo en común entre las dos pilas.
- Mientras ninguna de las pilas este vacía controlamos los topes de las mismas. Si el tope de ambas es el mismo, actualizamos el valor de la variable que almacena el último nodo en común con el tope de alguna, y desapilamos en las dos pilas. Si los topes de las dos son diferentes, detenemos el ciclo.
- Creamos una lista y agregamos al final el rótulo del último nodo en común.
-

- Mientras la pila1 no este vacía, desapilamos y agregamos al principio de la lista el rótulo del nodo desapilado.
- Mientras la pila2 no este vacía, desapilamos y agregamos al final de la lista el rótulo del nodo desapilado.

Finalmente tenemos en la lista el camino entre R1 y R2, la devolvemos y mostramos su contenido al usuario.

Ejercicio 9: Dados dos rótulos R1 y R2, devolver el ancestro común más cercano AC a R1 y R2 y dos listas L1 y L2 tal que L1 contiene el camino entre AC y R1 mientras que L2 contiene el camino entre AC y R2. Luego, mostrar el contenido de las listas L1 y L2 así como el rótulo de AC.

La solución de este problema es similar a la del ejercicio anterior. Se reciben como parámetros los rótulos de los dos nodos, y dos listas L1 y L2, donde en cada una estará el camino entre el ancestro común de ambos y el nodo correspondiente. Lo primero que hacemos es buscar los nodos cuyos rótulos son R1 y R2. Si alguno de los dos no pertenece al árbol, se informa a través de un cartel al usuario y el método no se ejecuta. Si ambos pertenecen al árbol se realiza lo siguiente:

- Creamos dos pilas auxiliares pila1 y pila2. En la pila1, mientras N1 sea distinto de la raíz, apilamos N1 y a este le asignamos su padre. Con la pila2 realizamos lo mismo, mientras N2 sea distinto de la raíz, apilamos N2 y a este le asignamos su padre. Al finalizar tenemos en cada una de las pilas los ancestros de cada nodo, excepto la raíz. Esta la asignamos a una variable especial donde almacenamos el último nodo en común entre las dos pilas.
- Mientras ninguna de las pilas este vacía controlamos los topes de las mismas. Si el tope de ambas es el mismo, actualizamos el valor de la variable que almacena el último nodo en común con el tope de alguna, y desapilamos en las dos pilas. Si los topes de las dos son diferentes, detenemos el ciclo.
- Cuando salimos del ciclo tenemos el ancestro común más cercano entre ambos nodos. Agregamos el ancestro común al final de ambas listas.
- Mientras la pila1 no este vacía, desapilamos y agregamos al final de la lista L1 el rotulo del nodo desapilado.
- Mientras la pila2 no este vacía, desapilamos y agregamos al final de la lista L2 el rotulo del nodo desapilado.

Finalmente, tenemos en L1 el camino entre AC y R1, y en L2 el camino entre AC y R2. Por último devolvemos el rotulo del ancestro común entre ambos nodos.

Ejercicio 10: Podar el árbol (de las hojas hacia la raíz) para que tenga una altura A determinada.

Antes de empezar a solucionar el problema verificamos lo siguiente: si la altura actual del árbol es menor o igual a A, se informa de la situación al usuario a través de un cartel y no se ejecuta el método. En cambio si la altura del árbol es mayor, la solución es la siguiente:

- Mientras la altura del árbol sea mayor a A:
  - Pedimos una lista con las hojas del árbol, es decir, los nodos externos, a través de un método auxiliar.
  - Por cada uno de los nodos almacenados en la lista, llamamos a removeNode(…) para que lo borre del árbol.

Al salir del ciclo, el árbol tiene la altura A que se requería.

Ejercicio 11: Dado el árbol cargado con las opciones 1 y 2, obtener un mapeo M con las entradas (R,p) donde R es el rótulo de un nodo y p la profundidad del nodo cuyo rótulo es R.

Los pasos para resolver este problema son los siguientes:

- Pedir el iterador de posiciones del árbol para tener todos los nodos.
- Crear un mapeo auxiliar de enteros/enteros.
- Por cada nodo n en el iterador, insertamos en el mapeo(con put(…)) el par (R,profundidad(n)), donde R es el rótulo de n y profundidad(n) es un método auxiliar que devuelve la profundidad del nodo n.

Al finalizar, en el mapeo tenemos almacenados pares con todos los rótulos de los nodos junto con su profundidad. Luego su contenido será mostrado al usuario.

Ejercicio 12: Dado el árbol cargado con las opciones 1 y 2, obtener un diccionario D con las entradas (p,R) donde p la profundidad del nodo cuyo rótulo es R.

Los pasos para resolver este problema son los siguientes

- Pedir el iterador de posiciones del árbol para tener todos los nodos.
- Crear un diccionario auxiliar de enteros/enteros.
- Por cada nodo n en el iterador, insertamos en el diccionario el par (profundidad(n),R), donde profundidad(n) es un

método auxiliar que calcula la profundidad del nodo n, y R es el rótulo de dicho nodo.

Al finalizar, en el diccionario se encuentran almacenados pares con todas las profundidades de los nodos del árbol junto con el rotulo del nodo correspondiente a dicha profundidad. Luego su contenido será mostrado al usuario.

Ejercicio 13: Dado un rótulo R, utilizar el mapeo M obtenido en (11) para mostrar la profundidad p del nodo que contiene a R como rótulo. Mostrar a continuación todos los rótulos de los nodos que tienen profundidad p utilizando el diccionario D obtenido en (11).

Se recibe como parámetro además del rotulo R, una lista L donde se devolverán los rótulos del resto de los nodos que tienen la misma profundidad que la del nodo cuyo rótulo es R. Para solucionar este problema, utilizamos el mapeo obtenido en el punto 11 y el diccionario obtenido en el punto 12. La solución es la siguiente:

- Pedir el mapeo del ejercicio 11.
- Pedir el diccionario del ejercicio 12
- Utilizando el método get (pasando el rótulo R como clave) del mapeo, obtenemos la profundidad p del nodo cuyo rótulo es R.
- Pedir al diccionario un iterable con todos los rótulos de los nodos del árbol que tienen profundidad p (utilizar el método findAll(…) pasando como clave p).
- Por cada rótulo del iterable, si es distinto a R(para no mostrar dos veces el mismo), agregamos el mismo al final de la lista L.

Al finalizar, en la lista tenemos todos los rótulos de los nodos que tienen la misma profundidad que el nodo cuyo rótulo es R. Devolvemos la profundidad p, y luego será mostrada al usuario así como también la lista.

Métodos auxiliares para resolver los ejercicios

- encontrarNodo(R): devuelve el nodo cuyo rótulo es R. Pide el iterador de posiciones del árbol y una vez que lo encuentra lo devuelve, si no lo encuentra retorna null.
- altura(N): calcula la altura del nodo N.
- profundidad(N): calcula la profundidad del nodo N.
- hojasDelArbol(): devuelve un iterable con los nodos externos del árbol. Crea una lista L, y pide el iterador de posiciones del árbol. Por cada nodo del iterador si es externo lo agrega a L. Al final devuelve la lista.