2014

Despliegue de Árbol Circular

SEGUNDO PROYECTO

KEVIN HERNÁNDEZ ROSTRÁN

Estructura de Datos

Instituto Tecnológico de Costa Rica

Introducción

El proyecto consiste en aplicar la Estructura de Datos denominada Árbol General, en la que cada uno de sus hijos puede tener una cantidad n de nodos. Los datos del árbol se cargan a partir de un archivo XML y para cargarlo correctamente y visualizar el árbol su raíz y sus nodos deben contener el TAG o etiqueta con el nombre "name" (sin comillas), y se obtiene el valor definido a esa etiqueta, se recorre el archivo y se almacena en el árbol general donde cada nodo tiene un hijo izquierdo y sus hermanos derechos, en la implementación se utilizan nodos enlazados entre sí. Se gráfica el árbol en formato de árbol circular (cada nivel es concéntrico con el nodo raíz),

El objetivo primordial es la posibilidad de dibujar diferentes elementos con respecto a coordenadas polares, donde cada nodo muestra su nombre y si se presiona click sobre el mismo se pliegan o despliegan cada uno de sus hijos y sub-hijos.

Para utilizar la librería pugiXML solamente hay que añadir al proyecto los archivos *pugixml.cpp*, *pugiconfig.hpp* y *pugixml.hpp* utilizando pugi::xml_document para manejar el documento y pugi::xml_node para manejar los nodos de la estructura del XML (más información: http://pugixml.googlecode.com/svn/tags/latest/docs/quickstart.html).

Las tecnologías implementadas son:

- Archivos en formato XML (para la lectura de nodos y su información).
- Librería pugiXML para leer archivos XML en C++.
- Ambiente de interfaces gráficas FLTK.
- Estructura de Árbol General (con nodos enlazados) de tipo hijo izquierdo/hermano derecho.

Descripción del problema

En un diagrama de "Árbol Circular" se dibujan tantos círculos concéntricos como niveles tenga el árbol, para ello se trabajan nodos hijos y sus padres, cada uno distribuido en cada nivel, para la distribución del árbol uno de sus problemas es la posición con respecto a las coordenadas en un plano, en que se deben traducir las coordenadas rectangulares en coordenadas polares.

Cada vez que se despliegan los nodos se deben redistribuir las posiciones del sus hijos, los hijos obtenidos del archivo XML se utiliza una representación basada en enlaces del tipo hijo izquierdo/hermano derecho.

Cada nodo se puede etiquetar con el nombre de algún atributo del nodo correspondiente. Un punto importante es que los nodos no se sobrepongan uno sobre el otro. Para resolver el problema la idea es crear límites por cada nodo padre en los que cada uno de sus nodos hijos estará en el gráfico.

Explicación de clases y rutinas principales

Clase Nodo <typename E>

La clase Nodo es para manejar los punteros o nodos hijosIzquierdo y hermanoDerecho del Árbol General almacenando:

- ✓ valor
- ✓ padre del nodo

- ✓ hijoIzquierdo del nodo
- ✓ hermanoDerecho del nodo
- ✓ Posición en X de la coordenada polar
- ✓ Posición en Y de la coordenada polar
- ✓ La visibilidad del nodo

Cuenta con tres punteros el padre, el hijoIzquierdo y el hermanoDerecho que en su construcción se les asigna un valor NULL, que en realidad más adelante se enlazará con otros nodos del árbol, el valor es de tipo E que significa el tipo de elemento asignado al árbol (ej. tipo string), las variables ejeX y ejeY almacenan la coordenada polar del nodo y su visibilidad al principio se define TRUE.

Clase Arbol <typename E>

Básicamente el Árbol inserta raíz, hijoIzquierdo, hermanoDerecho y elemento tipo <E> a cada uno de sus nodos y también retorna los valores como padre del nodo, hijoIzquierdo del nodo, o hermanoDerecho del nodo.

Para retornar valores de los nodos u hojas del árbol se utilizan los métodos:

- ✓ root (): conocer cuál es el nodo raíz del Árbol.
- ✓ padre (Nodo<E>* n): conocer cuál es el padre del Nodo que recibe como parámetro y lo retorna.
- ✓ hijoIzquierdo (Nodo<E>* n): conocer cuál es el hijoIzquierdo del Nodo que recibe como parámetro y lo retorna.
- ✓ hermanoDerecho(Nodo<E>* n): conocer cuál es el hermanoDerecho del Nodo que recibe como parámetro y lo retorna.

Para valores valores a los nodos u hojas del árbol se utilizan los métodos:

- ✓ insertaRaiz (E e): Recibe un valor del tipo de Árbol y se lo asigna a la raíz mantiene su padre, su hijoIzquierdo y su hermanoDerecho = NULL.
- ✓ insertaHijo (Nodo<E>* n, E e): recibe un Nodo y un elemento donde a ese nodo se le asigna un hijoIzquierdo y a ese hijoIzquierdo se le asigna el Elemento o dato.
- ✓ insertaHermano (Nodo<E>* n, E e): recibe un Nodo y un elemento donde a ese nodo se le asigna un hermanoDerecho y a ese hermanoDerecho se le asigna el Elemento o dato.

recorrido (Arbol<string>* arbol)

Algoritmo utilizado para recorrer el árbol que se ingresa como parámetro, el while recorre el nodo hasta que no encuentra hijoIzquierdo o hermanoDerecho así también con los demás nodos del Árbol mientras sus hijos o hermanos sean diferentes de NULL.

cargar_XML(const char* path)

Método que recorre el archivo que recibe como parámetro y lo almacena en el Árbol, al inicio se asignan los valores predeterminados de la cantidad de nodos del nivel a cero para que cada vez que se cargue un archivo se haga un “reset” de sus valores.

Variables utilizadas:

- ✓ pugi::xml_document doc: Para manejar el contenido del documento XML.
- ✓ if (!doc.load_file(path)) return -1: Verifica sí es un archivo XML

- ✓ pugi::xml_node: Útil para manejar los nodos del árbol del archivo XML.

La variable denominada comparar se utiliza para comparar el TAG con el string "name".

for (pugi::xml_node xmlRaiz = profile.first_child(); xmlRaiz; xmlRaiz = xmlRaiz.next_sibling())

El for aplica el mismo concepto de asignar un nodo y recorrer ese nodo hasta que nodo.next_sibling() no encuentre valores o nodos.

En cada loop del recorrido se evalúa o comparan etiquetas y se evalúa también que se inserte un hijoizq o hermanoderecho en el nivel#; si su hijoizquierdo es NULL le crea un hijo a ese nodo sino crea un hermanoDerecho y aumenta la Can_NIVEL# en uno para almacenar cantidad de hijos del nodo padre.

buscarNodo (string valor)

Método que busca un Nodo conforme al parámetro valor y lo retorna comparando el labelNodo con el parámetro valor

cambiarVisibleNodo(Nodo<string>* pnodo)

Recibe un pnodo como parámetro y evalúa si los hijos de ese nodo están visible si es true se cambia la visibilidad a FALSE sino a TRUE.

Evalúa:

If (nodo->visible==false) { nodo->visible = true; }else{ nodo->visible = false; }

Clase Lienzo de tipo Fl_Box

Objeto que dibuja el background de la ventana y las respectivas líneas del Árbol y si el árbol no está vacío en cada nodo se aplica el dibujo de líneas:

Dibuja (X,Y,X1,Y1) con:

- ✓ X= al ejeX del nodo padre
- ✓ Y= al ejeY del nodo padre
- ✓ X1= al ejeX del nodo
- ✓ Y1= al ejeY del nodo

Clase NodoBox de tipo Fl_Box

El constructor NodoBox(X,Y, W, H) dibuja un Fl_Box circular y aplica a su atributo

NodoBox->tooltip(valor), el valor del nodo->elemento o sea el elemento de cada nodo. Cada vez que se utiliza el método refrescarPantalla se redibujan cada uno de los nodos y sus respectivas ramas.

Cada vez que se genere un evento en el nodo se realiza lo siguiente:

- ✓ nodobuscado = buscarNodo (tooltip()): Busca el nodo según el valor del que se presionó.
- ✓ cambiarVisibleNodo (nodobuscado): Cambia los hijos del nodo de estado visible.
- ✓ refrescarPantalla (): Redibuja los nodos del Árbol.

refrescarPantalla ()

Al inicio realiza un clear() de la ventana, vuelve a llamar y crear nodos y ramas con las clases NodoBox y Lienzo aplicando las nuevas configuraciones por ejemplo, si se presiona un nodo cada uno de sus hijos se oculta y se redespliega el árbol.

Limpia la pantalla y refresca todos los elementos de la ventana evaluando si el nodo es oculto o no y refrescando y ocultando las ramas de cada nivel del árbol. Calcula en cada nodo su cardinalidad respectiva con la fórmula:

- CITA = (2*PI)/Cantidad de nodos del nivel.
- X = radio*cos(CITA*index del nodo).
- Y = radio*sin(CITA*index del nodo).

Descripción del formato de archivos

Variables para la cantidad de nodos hijos de cada Nivel:

- ✓ Can_NIVEL1
- ✓ Can_NIVEL2
- ✓ Can_NIVEL3
- ✓ Can_NIVEL4
- ✓ Can_NIVEL5

Variables para definir un nodo del árbol:

- ✓ Nodo<string>* aux1;
- ✓ Nodo<string>* aux2;
- ✓ Nodo<string>* aux3;
- ✓ Nodo<string>* aux4;
- ✓ Nodo<string>* aux5;

Variables a la hora de cargar los nodos del XML en el Árbol (tree):

- ✓ Nodo<string>* nodoRaiz
- ✓ Nodo<string>* nodoNivel1
- ✓ Nodo<string>* nodoNivel2
- ✓ Nodo<string>* nodoNivel3
- ✓ Nodo<string>* nodoNivel4
- ✓ Nodo<string>* nodoNivel5

Variables Globales:

- ✓ Arbol<string>* tree: Árbol general de todo el sistema
- ✓ Fl_Double_Window* win
- ✓ Lienzo* lienzo

Graficación de coordenadas:

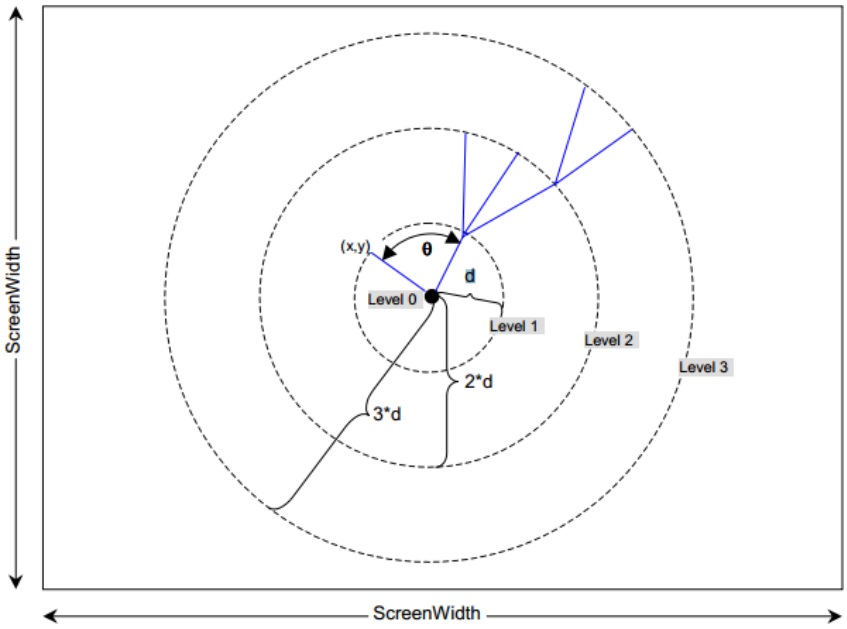

Figure 1 – General layout of radial graph

Tomado de: http://gbook.org/projects/RadialTreeGraph.pdf

Angulo = 2*pi/cantidad de nodos por nivel

nodoPosicionx = (d*NumerodeNivel) * sin(IndicedelNodo*Angulo))

nodoPosiciony = (d*NumerodeNivel) * cos(IndicedelNodo *Angulo))

Variables para calcular coordenadas polares:

- CITA = (2*PI)/Cantidad de nodos del nivel.
- X = radio*cos (CITA*index del nodo).
- Y = radio*sin (CITA*index del nodo).

Lo que almacena Nodo:

- Padre, hijo izquierdo, hermano derecho y visible (booleano).
- ejeX = resultado de X en la fórmula descrita anteriormente.
- ejeY = resultado de Y en la fórmula descrita anteriormente.

Análisis de resultados

Los objetivos del sistema fueron realizados al 90% debido a que la redistribución de los nodos cada vez que se pliegan o redespliegan los nodos no se vuelven a redistribuir el espacio entre los nodos, o sea cada nivel o círculo concéntrico está relacionado respecto al nodo raíz solamente y no con respecto a su padre.

Capacidades que no se desarrollaron:

- Repartir el espacio de la circunferencia entre los restantes nodos a la hora de plegar y desplegar nodos.
  - ✓ Solución:

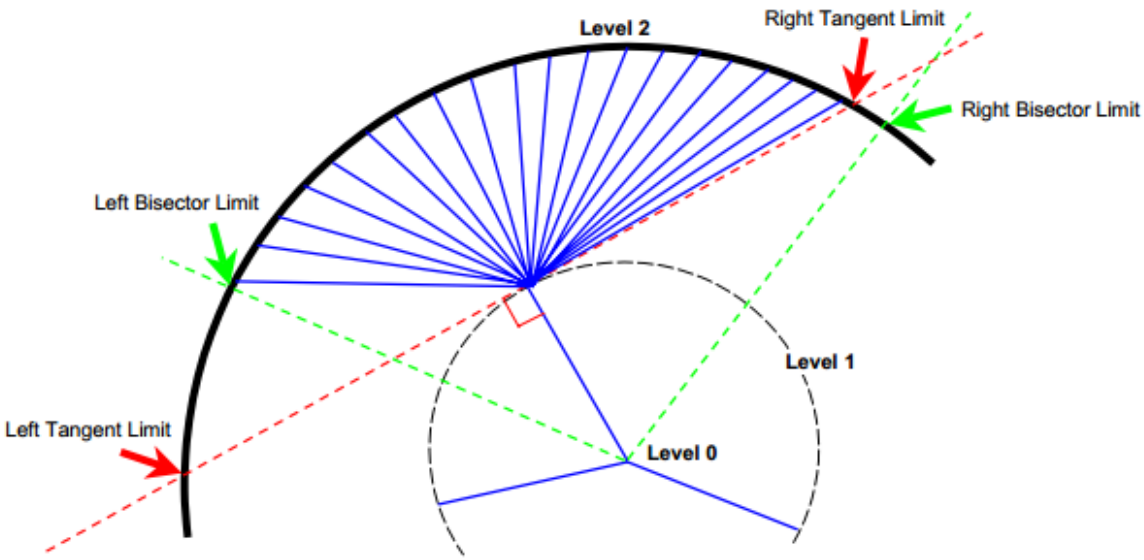

Figure 2 – Tangent and bisector limits for directories.

Tomado de: http://gbook.org/projects/RadialTreeGraph.pdf

La solución sería calcular los límites bisectores y limites tangenciales de cada nodo padre y sus respectivos hijos. Luego calcular el espacio angular del nodo o CITA:

CITA = (limite izquierdo - limite derecho) / # de nodos del Padre.

Por último recalcular los limites bisectores y tangenciales de el nodo siguiente y así aplicarlo a cada nodo de cada nivel (Nota: esta función es válida para los niveles mayores que Nivel 1).

Las tareas realizadas con éxito fueron:

- ✓ Carga de un archivo XML y el formato de sus nodos hijo-padre dependiendo del TAG utilizado para identificar la etiqueta del nodo del árbol.
- ✓ Despliegue de nodos hasta 5 niveles.
- ✓ Ocultamiento de nodos hijos.
- ✓ Distribución en coordenadas polares de cada nodo con respecto a la raíz.
- ✓ Cargar otro árbol y desplegarlo.
- ✓ Mostrar etiqueta del nodo.

Conclusiones

El proyecto contenía varias complejidades, pero se logró el cometido, solamente por la complejidad de redistribución de nodos que algunas ecuaciones de arco senos, tangentes y límites que bisecan a la amplitud del radio de cada uno de los nodos padres tenían una complejidad difícil de comprender, por lo que no se logró aplicar al proyecto. Por lo demás, cada nivel en sus ramas se diferencia por los colores de las mismas.

A lo largo de su desarrollo se fueron mostrando diferentes conflictos, al inicio la lectura del archivo XML fue complicado porque hay infinidad de librerías, pero la más sencilla para el manejo de nodos era pugixml.

Posteriormente la implementación de árbol general a través de enlaces significó un gran contratiempo por el hecho de ver cómo manejar los nodos, agregar hijos al árbol y después de ya conocer cómo llenar el Árbol, recorrerlo era otro desafío, de igual forma se implementa un algoritmo que recorre los niveles del árbol hasta encontrar un hijo o hermano que se encuentre en estado de NULL. Cuando ya se tenía el árbol lleno y el conocer cómo recorrerlo, graficarlo fue otro mundo de adversidades, debido a que la librería FLTK a la hora de aplicar un draw() se tenía que mantener la posición o el estado anterior almacenado en algún lugar porque todo, absolutamente todo el objeto se vuelve a calcular de nuevo o redibujar por ello se utilizan las variables ejeX y ejeY de la clase Nodo.

Otro punto complicado fue el hecho de aplicar las coordenadas polares a cada nodo y sus hijos, se logró en parte el objetivo, porque redistribuir el árbol el algoritmo era bastante complicado y modelarlo también por lo que se optó por utilizar como referencia el punto concéntrico de todo el diagrama o sea el nodo Raíz, cada coordenada se posiciona respecto a ese nodo.

Por último el plegar o desplegar nodos también fue una ardua tarea, pero se utilizó un método en el que se recorre el árbol al presionar click sobre el nodo y se busca ese nodo en el árbol y cuando se encuentra se utiliza otro método que toma todos los nodos hijos del nodo encontrado y les cambia de bandera (True o False).

El resultado fue satisfactorio y se logró la mayoría de los cometidos.