José Alberto Esquivel Patiño A01139626

Estructura de Datos

Tarea #6, 16 de Octubre, 2013

Ing. Román Martínez Martínez

Recursividad y Árboles Binarios

Resolución Problemas Recursivos

```
1.  //TAREA PARA EL JUEVES
2.  template <class T>
3.  int ABB<T>::contarNodosPadre(NodoArbol<T> *inicial)
4.  {
5.  if(inicial!=NULL)
6.  {
7.  if(inicial->izq != NULL || inicial->der != NULL)
8.          return 1+contarNodosPadre(inicial->izq)+contarNodosPadre(inicial->der);
9.  else
10.         return 0;
11. }else
12. {
13. return 0;
14. }
15. }
16.
17. template <class T>
18. void ABB<T>::desplegarArbol(NodoArbol<T> *inicial)
19. {
20.
21. if(inicial!=NULL)
22. {
23. if(inicial->izq != NULL || inicial->der != NULL)
24.         cout<<"Padre: "<<inicial->info<<endl;
25. else
```

```
26.         cout<<"Hoja: "<<inicial->info<<endl;
27. desplegarArbol(inicial->izq);
28. desplegarArbol(inicial->der);
29. }
30. }
31.
32. template <class T>
33. int ABB<T>::obtenerNivelRecursivamente(NodoArbol<T> *inicial)
34. {
35. int nivel_izquierdo, nivel_derecho;
36. if(inicial!=NULL)
37. {
38. nivel_izquierdo = obtenerNivelRecursivamente(inicial->izq);
39. nivel_derecho = obtenerNivelRecursivamente(inicial->der);
40. return (nivel_izquierdo > nivel_derecho) ? (nivel_izquierdo+1):(nivel_derecho+1);
41. }
42. else
43. {
44. return 0;
45. }
46. }
47.
48. template <class T>
49. int ABB<T>::contarNodos(NodoArbol<T> *inicial)
50. {
51. if(inicial!=NULL)
52. {
53. return 1+contarNodos(inicial->izq)+contarNodos(inicial->der);
54. }else
55. {
56. return 0;
57. }
58. }
59.
```

```
60. template <class T>
61. double ABB<T>::calcularPromedio(NodoArbol<T> *inicial, int numeroDeNodos)
62. {
63. if(inicial!=NULL)
64. {
65. return (inicial->info)*(1.00)/numeroDeNodos + calcularPromedio(inicial->izq,
    numeroDeNodos) +calcularPromedio(inicial->der, numeroDeNodos);
66.
67. }else
68. {
69. return 0;
70. }
71. }
72.
73. template <class T>
74. bool ABB<T>::buscarDato(NodoArbol<T> *inicial, T dato)
75. {
76. if(inicial!=NULL)
77. {
78. if(inicial->info == dato)
79. {
80.         return true;
81. }
82. else if(dato > inicial->info)
83. {
84.         return buscarDato(inicial->der, dato);
85. }
86. else
87. {
88.         return buscarDato(inicial->izq, dato);
89. }
90. }
91. else
92. {
```

```
93. return false;
94. }
95. 
96. }
97. 
98. template <class T>
99. void ABB<T>::insertarDatoRecursivamente(NodoArbol<T> *inicial, T dato)
100.    {
101.    if(inicial!=NULL)
102.    {
103.    if(inicial->info > dato)
104.    {
105.            if(inicial->izq != NULL)
106.            {
107.                    insertarDatoRecursivamente(inicial->izq, dato);
108.            }
109.            else
110.            {
111.                    NodoArbol<T> *nuevo = new NodoArbol<T>(dato);
112.                    inicial->izq = nuevo;
113.            }
114.    }
115.    else
116.    {
117.            if(inicial->der != NULL)
118.            {
119.                    insertarDatoRecursivamente(inicial->der, dato);
120.            }
121.            else
122.            {
123.                    NodoArbol<T> *nuevo = new NodoArbol<T>(dato);
124.                    inicial->der = nuevo;
125.            }
126.    }
```

```
127.    }
128.    else
129.    {
130.    NodoArbol<T> *nuevo = new NodoArbol<T>(dato);
131.    raiz = nuevo;
132.    }
133.    }
134.    template <class T>
135.    void ABB<T>::desplegarArbolNivelPorNivel()
136.    {
137.    queue< NodoArbol<T>* > fila;
138.    fila.push(raiz);
139.    while(!fila.empty())
140.    {
141.            NodoArbol<T> *temporal = fila.front();
142.            fila.pop();
143.            cout<<temporal->info<<endl;
144.            if(temporal->izq!=NULL)
145.                    fila.push(temporal->izq);
146.            if(temporal->der!=NULL)
147.                    fila.push(temporal->der);
148.    }
149.    }
150.    //FIN DE TAREA PARA EL JUEVES
```

Modificación de Clase Árbol y Programa Aplicación

```
1.  #include <iostream>
2.  #include <string>
3.  #include <fstream>
4.
5.  using namespace std;
6.
7.  template <class T>
8.  class NodoArbol
```

```
{ public:
        T info;
        int repeticiones;
        NodoArbol<T> *izq, *der;
        NodoArbol() { izq = der = NULL; }
        NodoArbol(T dato){ info = dato; repeticiones = 0; izq = der = NULL; }
};

template <class T>
class ABB
{
private:
                NodoArbol<T> *raiz;
public:
                ABB() { raiz = NULL; }
                NodoArbol<T>* getRaiz(){return raiz;}
                void insertarDatoRecursivamente(NodoArbol<T> *inicial, T dato);
                void desplegarArbol(NodoArbol<T> *inicial);
                ~ABB() { libera(raiz); }
};

template <class T>
void libera (NodoArbol<T>* raiz)
{ //Observar que al ser recursive, es una función libre llamada por el método
        if (raiz != NULL)
        {       libera(raiz->izq);
                libera(raiz->der);
                delete(raiz);
        }
}

template <class T>
void ABB<T>::insertarDatoRecursivamente(NodoArbol<T> *inicial, T dato)
{
```

```
43.         if(inicial!=NULL)
44.         {
45.                 if(inicial->info == dato)
46.                 {
47.                         inicial->repeticiones += 1;
48.
49.                 }
50.                 else if(dato < inicial->info)
51.                 {
52.                         if(inicial->izq != NULL)
53.                         {
54.                                 insertarDatoRecursivamente(inicial->izq, dato);
55.                         }
56.                         else
57.                         {
58.                                 NodoArbol<T> *nuevo = new NodoArbol<T>(dato);
59.                                 inicial->izq = nuevo;
60.                         }
61.
62.                 }
63.                 else
64.                 {
65.                         if(inicial->der != NULL)
66.                         {
67.                                 insertarDatoRecursivamente(inicial->der, dato);
68.                         }
69.                         else
70.                         {
71.                                 NodoArbol<T> *nuevo = new NodoArbol<T>(dato);
72.                                 inicial->der = nuevo;
73.                         }
74.                 }
75.         }
76.         else
```

```
77.         {
78.                 NodoArbol<T> *nuevo = new NodoArbol<T>(dato);
79.                 raiz = nuevo;
80.         }
81. }
82.
83. template <class T>
84. void ABB<T>::desplegarArbol(NodoArbol<T> *inicial)
85. {
86.
87.         if(inicial!=NULL)
88.         {
89.                 desplegarArbol(inicial->izq);
90.                 if(inicial->repeticiones == 0)
91.                 {
92.                         cout<<inicial->info<<endl;
93.                 }
94.                 else
95.                 {
96.                         for(int i = 0; i<inicial->repeticiones; i++)
97.                                 cout<<inicial->info<<' ';
98.                         cout<<endl;
99.                 }
100.                desplegarArbol(inicial->der);
101.        }
102.  }
103.  int main()
104.  {
105.          ABB<int> arbol;
106.          ifstream arch;
107.          ofstream archivoSalida;
108.          string nomarch;
109.          int dato;
110.          cout << "Ingrese el nombre del archivo a cargar: "<<endl;
```

```
111.         cin >> nomarch;
112.
113.         arch.open(nomarch.c_str());
114.         while (!arch.eof())
115.         {
116.                 arch >> dato;
117.                 arbol.insertarDatoRecursivamente(arbol.getRaiz(), dato);
118.         }
119.         arch.close();
120.
121.         cout<<"Este es el arbol guardado en el archivo: "<<endl;
122.         arbol.desplegarArbol(arbol.getRaiz());
123.
124.         int decision;
125.         cout<<"\nDesea insertar mas datos al arbol? \n1)Si \n2) No"<<endl;
126.         cin>>decision;
127.         archivoSalida.open(nomarch.c_str(), ios::app);
128.         while(decision==1)
129.         {
130.                 cout<<"Inserte un dato entero:"<<endl;
131.                 cin>>dato;
132.
133.                 archivoSalida<<dato<<endl;
134.                 arbol.insertarDatoRecursivamente(arbol.getRaiz(), dato);
135.
136.                 cout<<"\nDesea insertar mas datos al arbol? \n1)Si
    \n2)No"<<endl;
137.                 cin>>decision;
138.         }
139.         archivoSalida.close();
140.
141.         cout<<"Este es el arbol modificado guardado en el archivo: "<<endl;
142.         arbol.desplegarArbol(arbol.getRaiz());
143.   }
```

Investigación

A continuación se presentan los tipos de árboles binarios encontrados en la investigación rápida que realicé:

- Rooted Binary Tree: Árbol binario que tiene una raíz y donde cada nodo tiene como máximo 2 hijos.
- Full Binary Tree: (Proper Binary Tree, 2-Tree, Strictly Binary Tree) Todos los nodos excepto las hojas, tienen exactamente 2 hijos. Las hojas tienen 0 hijos.
- Perfect Binary Tree: Es un Full Binary Tree, en donde todas las hojas están en el mismo nivel y cada padre tiene dos hijos.
- Complete Binary Tree: Es un árbol binario en donde cada nivel, excepto el último, está completamente lleno, y los nodos son puestos lo más lejos hacia la izquierda que se pueda. Se llama *almost complete binary tree* cuando el último nivel no está completamente lleno. A esto se le denomina un *heap.*
- Infinite Complete Binary Tree: Es un árbol con un número infinito de niveles. En donde cada nodo tiene dos hijos. El conjunto de todos los nodos es contable e infinito, pero el set de los caminos que hay es incontable e infinito.
- Balanced Binary Tree: Es un árbol binario en donde la profundidad de los subárboles izquierdo y derecho de cada nodo difieren en 1 o menos.
- Degenerate Tree: Árbol donde para cada árbol o nodo padre, hay solamente un hijo hacia algun lado, como el lado izquierdo.

Adicional:

Un heap es una estructura de datos basada en un árbol, donde los datos son introducidos de arriba a abajo, de izquierda a derecha. No te puedes mover a un nivel sin haber llenado el nivel anterior.