UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

Tecnicatura en diseño y programación de videojuegos

UNLVIRTUAL

Programación de videojuegos I

Unidad 3
Árboles

Docentes
Germán Gaona
Mauro Walter

CONTENIDOS

INTRODUCCIÓN

La noción de árbol se encuentra frecuentemente inmersa en la vida diaria y, en general, implica cierta estructura jerárquica. Así, por ejemplo, la mejor manera de reconstruir nuestra historia familiar es haciendo un árbol genealógico. También podemos recurrir a organigramas si deseamos explicitar la cadena formal de mando en una organización o representar ciertos procesos de análisis como los árboles de decisión, en los que vamos probando distintos caminos posibles o marcando ocurrencia de eventos.

Árbol
Abstracción matemática, utilizada principalmente para describir propiedades dinámicas de los algoritmos.

Un *árbol* es una abstracción matemática, utilizada principalmente para describir propiedades dinámicas de los algoritmos, las cuales son representadas como una estructura abstracta de datos. Como los árboles son aplicados a una gran variedad de problemas, existen diferentes tipos de árboles específicos para cada situación.

3.1. ÁRBOLES

La definición de árbol más amplia es conocida como *árbol libre*. Un árbol es un conjunto no vacío de vértices y arcos que debe verificar ciertos requisitos. El vértice o nodo es un elemento simple y puede contener datos asociados; un arco es un enlace entre dos vértices, y un camino es una lista sucesiva de vértices distintos conectados por arcos.

Con estas definiciones podemos establecer que un árbol sólo contiene un camino posible entre dos nodos cualesquiera.

Veamos otras definiciones:

Árbol desordenado: árbol con raíz, o simplemente árbol, es similar al anterior, pero -en este caso- designa a uno de los vértices como raíz. Cada vértice se considera raíz de un subárbol, siendo ésta una definición recursiva.

De esta definición se desprenden algunos conceptos:

Nodo padre: es aquel que tiene cierto número de nodos subordinados llamados hijos.

Nodo hijo: es aquel que depende de un nodo padre.

Nodo raíz: es aquel que no tiene nodos padres.

Nodo hoja: es aquel que no posee hijos.

Nodo intermedio: es aquel que posee nodos hijos y a su vez está subordinado a un padre.

Árbol ordenado: a diferencia del anterior, en este tipo de árboles, los nodos o vértices se encuentran ordenados por algún criterio.

Árboles binarios: son árboles donde cada uno de sus nodos apunta solamente a otros dos nodos, uno izquierdo y otro derecho.

Árboles n-arios: son árboles ordenados donde cada nodo tiene exactamente *n* hijos. Es necesario, entonces, definir nodos especiales que se encuentren vacíos, los cuales no tendrán ningún hijo. A estos nodos los llamamos *nodos externos.*

Árbol libre

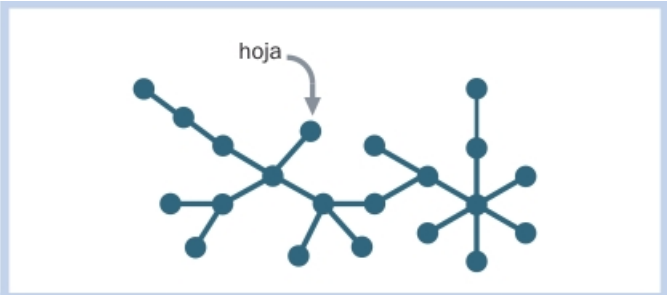

Fig. 1

Árbol desordenado (con raíz)

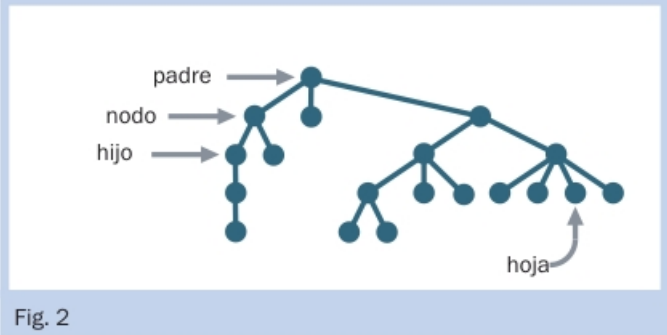

Fig. 2

Árbol binario

Fig. 3

3.1.1. Moverse a través de un árbol

Las operaciones de un árbol se encuentran especializadas de acuerdo al tipo de estructura que estemos usando. Sin embargo, se pueden establecer algunos criterios importantes a tener en cuenta cuando operamos con árboles en general.

Antes de continuar con este tema, dejaremos en claro algunos conceptos necesarios para un desarrollo posterior.

Nivel (o profundidad): es una medida cuantitativa que indica la distancia existente entre la raíz y el nodo en cuestión. Por convención, el nivel de la raíz es *0* y va incrementando a medida que encontramos una ramificación.

Altura: es el máximo nivel encontrado dentro de un árbol o subárbol.

Grado: es una medida cuantitativa que indica el número de descendientes directos de un nodo. También podemos identificar el grado de un árbol, que es simplemente el máximo grado de un nodo.

Dado el siguiente árbol, obtendremos los distintos valores de altura, nivel y grado de cada nodo:

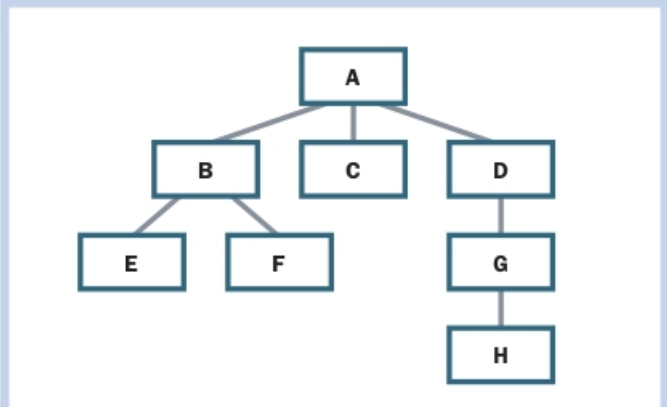

Fig. 4

Nodo	Nivel	Altura	Grado
A	0	3	3
B	1	1	2
C	1	0	0
D	1	2	1
E	2	0	0
F	2	0	0
G	2	1	1
H	3	0	0

3.1.2. Recorridos

Recorridos de un árbol
Consisten en ir visitando los nodos de un árbol, siguiendo alguna estrategia particular, pero siempre teniendo en cuenta los vínculos existentes entre los nodos. Existen dos tipos básicos de recorrido: en profundidad y en amplitud.

Consisten en ir visitando los nodos de un árbol, siguiendo alguna estrategia particular, pero siempre teniendo en cuenta los vínculos existentes entre los nodos. Existen dos tipos básicos de recorrido, que dependen del orden en que se visitan los nodos: en profundidad y en amplitud.

Recorrido en profundidad

Consiste en adentrarse en cada nodo, llegando al mayor nivel posible antes de pasar a la rama contigua, y repetir así el proceso hasta cumplir con la condición de fin. El algoritmo utilizado es de naturaleza netamente recursiva.

Existen subclasificaciones de este tipo de recorrido, dependiendo del momento en que se procesa cada nodo:

- Orden previo (o pre-orden): se procesa primero el nodo raíz en cuestión y luego todas sus ramas.
- Orden simétrico (o in-orden): se procesa, primero, una o más ramas; luego. el nodo raíz, y por último, todas las ramas restantes.
- Orden posterior (o post-orden): primero se procesan todas las ramas y, por último, el nodo raíz.

Recorrido en amplitud

Consiste en hacer un barrido por capas en el árbol, recorriendo nivel por nivel hasta finalizar. El algoritmo que describe este proceso es de naturaleza iterativa.

3.2. ÁRBOLES BINARIOS DE BÚSQUEDA (ABB)

Se trata de árboles de orden igual a dos, en los que se cumple que, para cada nodo, los elementos del subárbol izquierdo son menores que el elemento del nodo, mientras que los elementos del subárbol derecho son mayores. La estructura del nodo consiste en tres partes: una para el dato, otra para el link izquierdo y la última para el link derecho.

Árboles binarios de búsqueda (ABB)
Se trata de árboles de orden igual a dos, en los que se cumple que, para cada nodo, los elementos del subárbol izquierdo son menores que el elemento del nodo, mientras que los elementos del subárbol derecho son mayores. La estructura del nodo consiste en tres partes: una para el dato, otra para el link izquierdo y la última para el link derecho.

Gráfico de ejemplo:

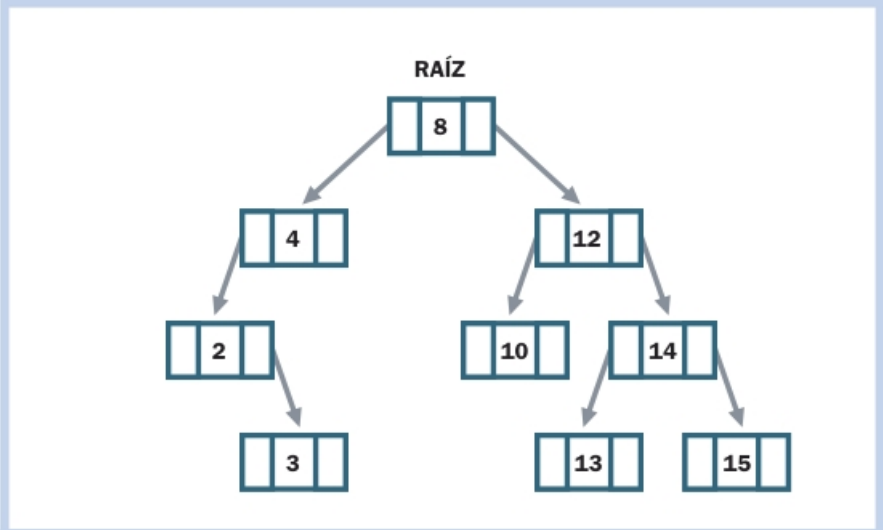

Fig. 5

```
class ArbolABB {
  private:

   // Clase local de Lista para Nodo de ArbolBinario:
   class Nodo {
     public:
      // Constructor:
      Nodo(const int dat, Nodo *izq=NULL, Nodo *der=NULL) :
        dato(dat), izquierdo(izq), derecho(der) {}

      // Miembros:
      int dato;
      Nodo *izquierdo;
      Nodo *derecho;
   };

   // Punteros de la lista, cabeza y nodo actual:
   Nodo *raiz;
   Nodo *actual;
   int contador;
   int altura;

  public:
   // Constructor y destructor
   ArbolABB() : raiz(NULL), actual(NULL) {}
   ~ArbolABB() { }

   // Comprobar si el árbol está vacío
   bool Vacio(Nodo *r) { return r==NULL; }

   // Comprobar si es un nodo hoja
   bool EsHoja(Nodo *r) { return !r->derecho && !r->izquierdo; }
  };
```

3.2.1. Operaciones

El conjunto de operaciones que se pueden hacer sobre un ABB es similar al que se realiza sobre otras estructuras de datos como las listas o alguna otra propia de los árboles.

Operaciones sobre un ABB
Son tres y se denominan Búsqueda, inserción y eliminación.

Búsqueda

Partiendo siempre del nodo raíz, el modo de buscar un elemento se define de forma recursiva.

1. Si el árbol está vacío, terminamos la búsqueda: el elemento no está en el árbol.

2. Si el valor del nodo raíz es igual al del elemento que buscamos, terminamos la búsqueda con éxito.

3. Si el valor del nodo raíz es mayor que el elemento que buscamos, continuaremos la búsqueda en el árbol izquierdo.

4. Si el valor del nodo raíz es menor que el elemento que buscamos, continuaremos la búsqueda en el árbol derecho.

```
bool ArbolABB::Buscar(const int dat) {
   actual = raiz;

   // Todavía puede aparecer, ya que quedan nodos por inspeccionar
   while(!Vacio(actual)) {
      if(dat == actual->dato) return true; // entero encontrado
      else if(dat > actual->dato) actual = actual->derecho; // Seguir
      else if(dat < actual->dato) actual = actual->izquierdo;
   }
   return false; // No está en el árbol
}
```

El valor de retorno de una función de búsqueda en un ABB puede ser un puntero al nodo encontrado, o NULL, si no se ha encontrado.

Inserción

Para insertar un elemento es necesario remitirse al algoritmo de búsqueda. Si el elemento está en el árbol, no se insertará. Si no lo está, lo hará a continuación del último nodo visitado.

Para conservar una referencia al padre del nodo raíz *actual,* se necesita un puntero auxiliar cuyo valor inicial sea NULL.

1. *Padre* es igual a NULL y *nodo* es igual a *raíz.*

2. Bucle: mientras *actual* no sea un árbol vacío, o hasta que se encuentre el elemento, podemos hacer lo siguiente:

   2.1. Si el valor del nodo raíz es mayor que el elemento que se busca, se continúa la búsqueda en el árbol izquierdo: *padre* es igual a *nodo* y *nodo* es igual a su hijo izquierdo.

   2.2. Si el valor del nodo raíz es menor que el elemento que se busca, se continúa la búsqueda en el árbol derecho: *padre* es igual a *nodo* y *nodo* es igual a su hijo derecho.

3. Si *nodo* no es NULL, el elemento está en el árbol. Por lo tanto, no se inserta y termina el algoritmo.

4. Si *padre* es NULL, el árbol está vacío. Entonces, el nuevo árbol sólo contendrá el nuevo elemento, que será la raíz del árbol.

5. Si el elemento es menor que el padre, entonces se inserta el nuevo elemento como un nuevo árbol izquierdo de *padre.*

6. Si el elemento es mayor que el padre, entonces se inserta el nuevo elemento como un nuevo árbol derecho de *padre.*

```
void ArbolABB::Insertar(const int dat) {
   Nodo *padre = NULL;

   actual = raiz;
   // Buscar el entero en el árbol, manteniendo un puntero al nodo padre
   while(!Vacio(actual) && dat != actual->dato) {
      padre = actual;
      if(dat > actual->dato) actual = actual->derecho;
      else if(dat < actual->dato) actual = actual->izquierdo;
   }

   // Si se ha encontrado el elemento, regresar sin insertar
   if(!Vacio(actual)) return;

   // Si padre es NULL, entonces el árbol estaba vacío, el nuevo nodo será
   // el nodo raiz
   if(Vacio(padre)) raiz = new Nodo(dat);

   // Si el entero es menor que el que contiene el nodo padre, lo insertamos
   // en la rama izquierda
   else if(dat < padre->dato) padre->izquierdo = new Nodo(dat);

   // Si el entero es mayor que el que contiene el nodo padre, lo insertamos
   // en la rama derecha
   else if(dat > padre->dato) padre->derecho = new Nodo(dat);
}
```

Eliminación

Para borrar un elemento también se utiliza como ayuda el algoritmo de búsqueda. Si el elemento no está en el árbol, no se puede borrar. Pero si está, hay dos casos posibles:

- Se trata de un nodo *hoja:* se borra directamente.
- Se trata de un nodo *rama:* no se puede eliminar, puesto que se perderían todos los elementos del árbol de los que el nodo *actual* es padre. En su lugar, se busca el nodo ubicado más a la izquierda del subárbol derecho, o el que está más a la derecha del subárbol izquierdo, y se intercambian sus valores. A continuación, se elimina el nodo *hoja.*

1. Se utiliza un puntero auxiliar para conservar una referencia al padre del nodo raíz actual. El valor inicial para ese puntero es NULL. Padre = NULL
2. Si el árbol está vacío: el elemento no está en el árbol; por lo tanto, el algoritmo termina sin eliminar ningún elemento.
3. Si el valor del nodo raíz es igual al del elemento que se está buscando, estamos ante uno de los siguientes casos:

   3.1. El nodo raíz es un nodo hoja:

      3.1.1. Si *padre* es NULL, el nodo raíz es el único del árbol; por lo tanto, el puntero al árbol debe ser NULL.

      3.1.2. Si raíz es la rama derecha de *padre,* esa rama apuntará a NULL.

      3.1.3. Si raíz es la rama izquierda de *padre,* esa rama apuntará a NULL.

      3.1.4. Se elimina el nodo y el algoritmo termina.

   3.2. El nodo no es un nodo hoja:

      3.2.1. Se busca el *nodo* más a la izquierda del árbol derecho de raíz o el más a la derecha del árbol izquierdo. Debemos tener en cuenta que puede ocurrir que sólo exista uno de esos árboles. Al mismo tiempo, se actualiza *padre* para que apunte al padre de *nodo.*

3.2.2. Se intercambian los elementos de los nodos raíz y *nodo.*

3.2.3. Se borra el nodo *nodo.* Esto significa volver al punto 3, ya que puede suceder que *nodo* no sea un nodo hoja.

4. Si el valor del nodo raíz es mayor que el elemento que se busca, se continúa la búsqueda en el árbol izquierdo.

5. Si el valor del nodo raíz es menor que el elemento que se busca, se continúa la búsqueda en el árbol derecho.

```
void ArbolABB::Borrar(const int dat){
   Nodo *padre = NULL;
   Nodo *nodo;
   int aux;

   actual = raiz;
   // Mientras sea posible que el valor esté en el árbol
   while(!Vacio(actual)) {
      if(dat == actual->dato) { // Si el valor está en el nodo actual
         if(EsHoja(actual)) { // Y si además es un nodo hoja: lo borramos
            if(padre) // Si tiene padre (no es el nodo raíz)
               // Anulamos el puntero que le hace referencia
               if(padre->derecho == actual) padre->derecho = NULL;
               else if(padre->izquierdo == actual) padre->izquierdo = NULL;
            delete actual; // Borrar el nodo
            actual = NULL;
            return;
         }
         else { // Si el valor está en el nodo actual, pero no es hoja
            // Buscar nodo
            padre = actual;

            // Buscar nodo más izquierdo de rama derecha
            if(actual->derecho) {
               nodo = actual->derecho;
               while(nodo->izquierdo) {
                  padre = nodo;
                  nodo = nodo->izquierdo;
               }
            }

            // O buscar nodo más derecho de rama izquierda
            else {
               nodo = actual->izquierdo;
               while(nodo->derecho) {
                  padre = nodo;
                  nodo = nodo->derecho;
               }
            }

            // Intercambiar valores de nodo a borrar u nodo encontrado
            // y continuar, cerrando el bucle. El nodo encontrado no tiene
            // por qué ser un nodo hoja, cerrando el bucle nos aseguramos
            // de que sólo se eliminan nodos hoja.
            aux = actual->dato;
            actual->dato = nodo->dato;
            nodo->dato = aux;
            actual = nodo;
         }
      }
      else { // Todavía no hemos encontrado el valor, se continúa buscando
         padre = actual;
         if(dat > actual->dato) actual = actual->derecho;
         else if(dat < actual->dato) actual = actual->izquierdo;
      }
   }
}
```

Eliminación de un nodo hoja:

En este ejemplo, se desea eliminar el elemento 3.

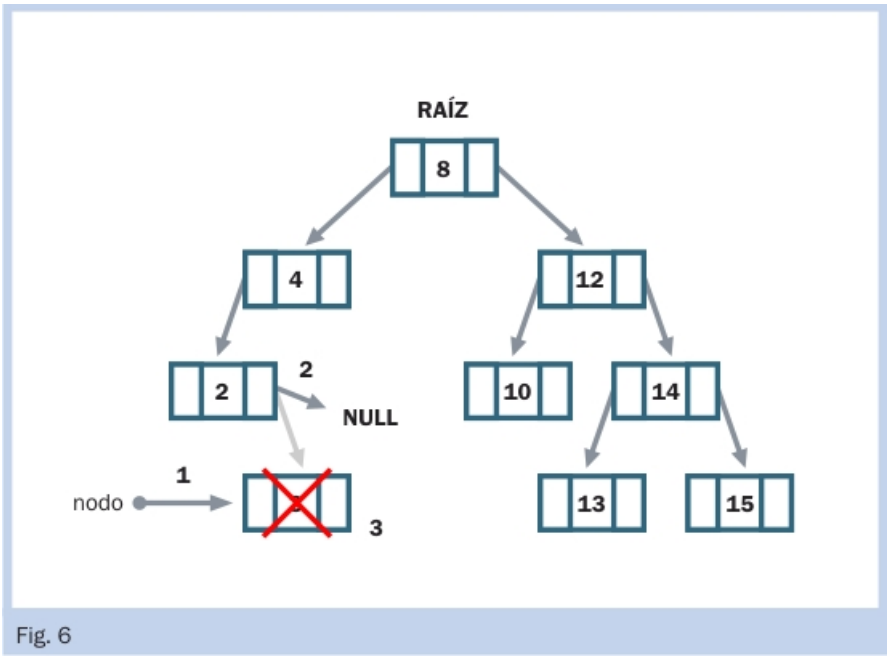

Fig. 6

Eliminación de un nodo rama con intercambio de un nodo hoja:

En este ejemplo, se desea eliminar el elemento 4.

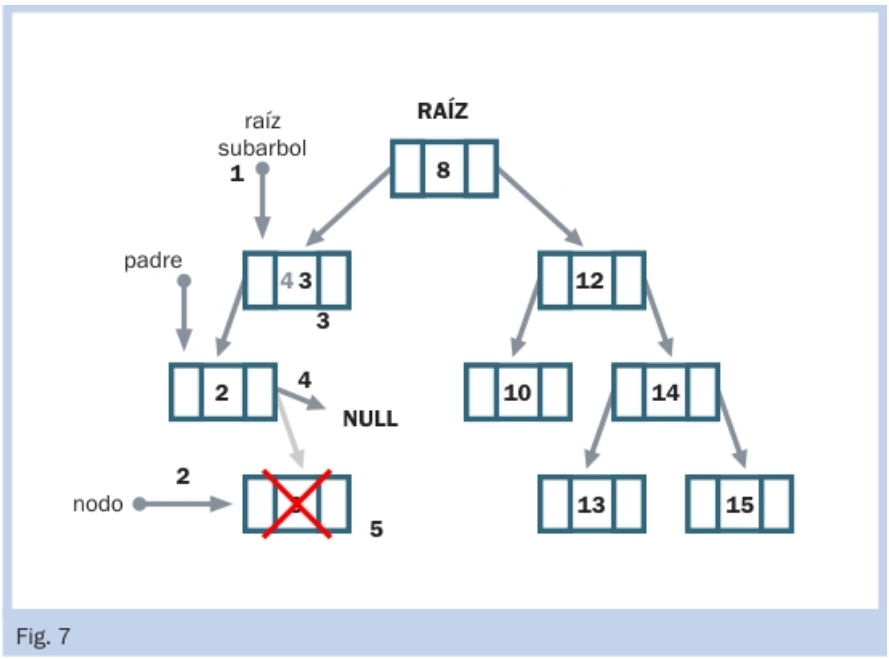

Fig. 7

Eliminación de un nodo rama con intercambio de un nodo rama:

En este ejemplo, se desea eliminar el elemento 6.

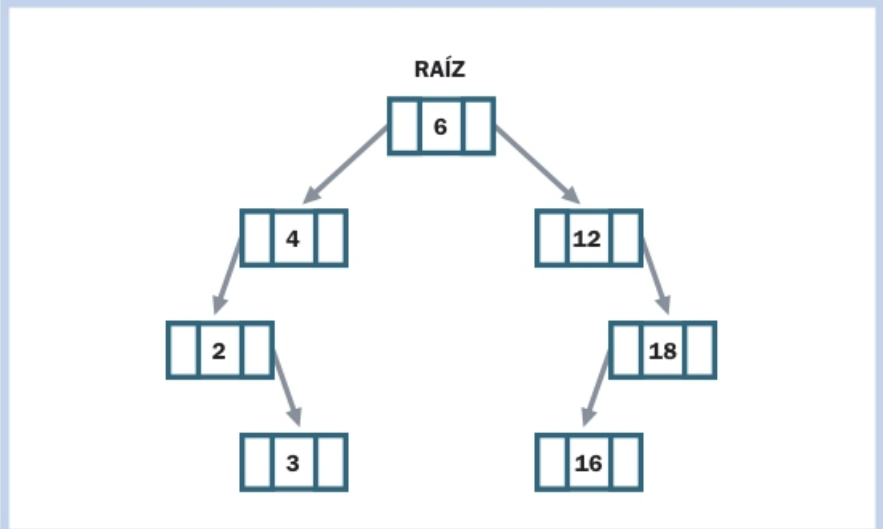

Fig. 8

Paso 1:

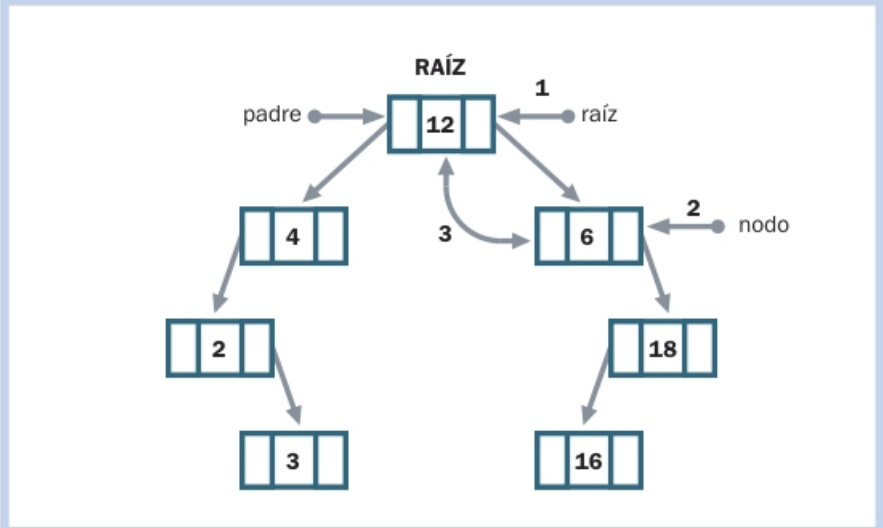

Fig. 9

Ahora, se debe repetir el bucle para el nodo 6 nuevamente, ya que no podemos eliminarlo.

Paso 2:

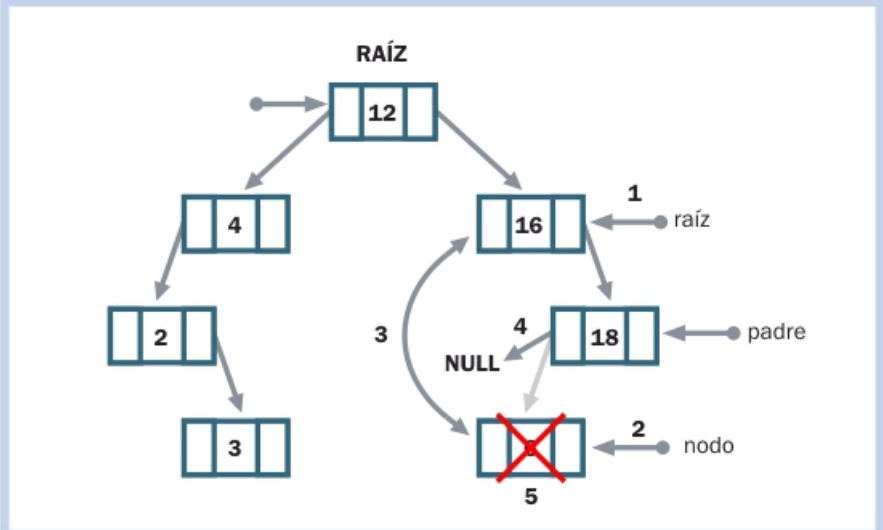

Fig. 10

3.2.2. Aplicación de árboles binarios en videojuegos

En este apartado, analizaremos la creación de un juego de preguntas y respuestas, según el cual el usuario piensa en cualquier objeto de la realidad, el juego va realizando ciertas preguntas basándose en su conocimiento e intentando adivinar cuál es el objeto. Si el objeto no es adivinado, es decir, no se encuentra registrado por la computadora, deberá agregarse al conjunto de objetos conocidos en un orden determinado.

Para este juego se utilizará un árbol binario que almacena el conjunto de objetos y preguntas que conforman el *conocimiento* de la computadora. Los nodos hojas contienen descripción de objetos y los nodos padres, preguntas. Además, estos últimos poseen una referencia a un nodo *si* y a otro nodo *no.*

Gráfico de ejemplo:

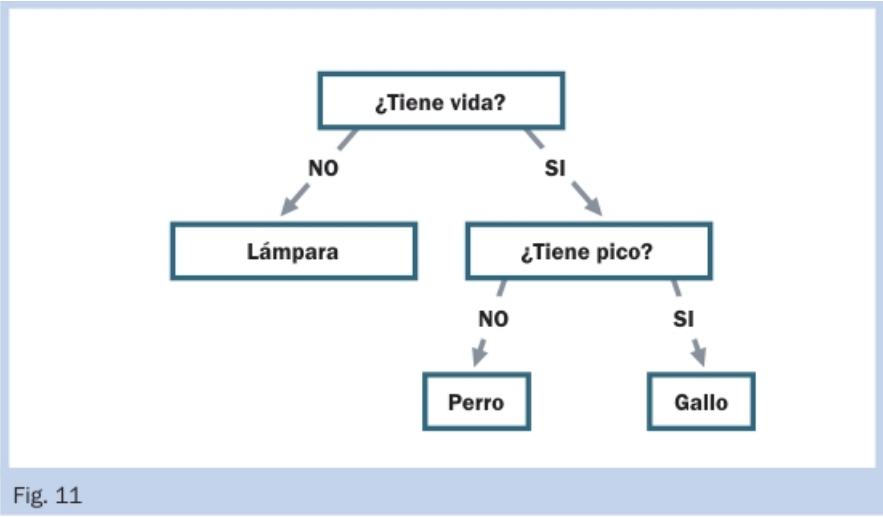

Fig. 11

Al comenzar el juego, si la respuesta a la pregunta es *sí,* se busca el nodo que hace referencia al enlace *si.* De lo contrario, si la respuesta es *no,* se busca el nodo que hace referencia al enlace *no.* Si el nodo no es hoja, se realiza la pregunta alojada en el nodo actual, pero si lo es, el juego preguntará si es el objeto registrado en ese nodo. Ahora bien, si el jugador responde que *sí,* la computadora ha adivinado el objeto y resulta ganadora. Pero si el jugador responde que *no,* él será quien habrá ganado y deberá

escribir cuál fue el objeto pensado, junto con una descripción asociada, permitiendo así que el juego incremente su conocimiento.

Ejemplo de una partida:

```
¡Hola! Bienvenido al juego de las preguntas.
Piensa en un objeto…

Es un animal? s
Estas pensando en un pez?
Presiona ‘s’ por si, o ‘n’ por no: n
Ganaste!

En qué objeto estabas pensando?: gato
Por favor, ingresa una pregunta que distinga un pez de un gato: Es un mamífero?
Si pensaras en un gato, cuál sería la respuesta a esta pregunta?: s
Gracias! Has expandido mi conocimiento!

Quieres jugar otra vez?: s

Es un animal? s
Es un mamífero? s
Estás pensando en un gato?
Presiona ‘s’ por sí, o ‘n’ por no: s
He ganado!
```

Ahora, la siguiente clase *ABNodo* refleja la estructura básica de un nodo, necesaria para el juego:

```
class ABNodo {
private:
        string mensaje; //Almacena la pregunta, o si es nodo hoja, un objeto
        ABNodo* nodoNo; //Enlace a nodo por respuesta "no"
        ABNodo* nodoSi; //Enlace a nodo por respuesta "si"
public:
        ABNodo(string mensajeNodo) { //Constructor
                mensaje = mensajeNodo;
                nodoNo = NULL;
                nodoSi = NULL;
        }

        // Devuelve si el nodo es hoja, por lo tanto no es pregunta
        bool esPregunta() {
                if (nodoNo == NULL && nodoSi == NULL)
                        return false;
                else
                        return true;
        }
        void consultar();
        void actualizarArbol();
};
```

Como puede verse en el método *esPregunta,* cuando un nodo tiene sus dos enlaces en NULL, la computadora no tiene más preguntas por hacer y está lista para suponer qué objeto es.

También debemos definir una clase *ArbolB,* que es nuestro Árbol binario que aloja los nodos de tipo *ABNodo*.

```
class ArbolB{
private:
        ABNodo* nodoRaiz;
public:
        ArbolB(string pregunta, string suposicionSi, string suposicionNo){
                nodoRaiz = new ABNodo(pregunta);
                nodoRaiz->setNodoSi(new ABNodo(suposicionSi));
                nodoRaiz->setNodoNo(new ABNodo(suposicionNo));
        }

        void consultar(){
                nodoRaiz->consultar();
        }
};
```

Cuando se crea un nuevo juego, se necesita una referencia al nodo raíz, el cual mantiene la primera pregunta del juego, con sus correspondientes enlaces SI y NO a los nodos hoja.

A continuación, se explica el funcionamiento lógico del juego.

Por razones de simplicidad, no hemos definido el cuerpo del método *consultar* en *ABNodo*. Pero, básicamente, el mismo realiza las siguientes acciones: hace la pregunta correspondiente, siempre y cuando sea un nodo padre; si el jugador responde que *sí,* la computadora hará la siguiente pregunta con *nodoSi->consultar(),* mostrando en pantalla *mensaje*, y *nodoNo->consultar()* si el jugador responde que *no.*

Cuando es un nodo hoja, dispara la suposición mostrando en pantalla *mensaje*. Si adivina el objeto, termina el juego, pero si no, se llama al método *actualizarArbol(),* donde el usuario escribe cuál fue el objeto que pensó y una característica que distinga al objeto pensado del supuesto por la computadora.

Esta actualización le servirá al juego para las siguientes partidas y consiste en agregar nuevos nodos al árbol de la siguiente manera:

```
cout << "En que objeto estabas pensando?:" << endl;

//El usuario ingresa su objeto
string objetoUsuario;
cin >> objetoUsuario;

cout << "Por favor, ingresa una pregunta que distinga un " << mensaje << "de un " <<
objetoUsuario << ": " << endl;

//El usuario ingresa la pregunta asociada
string preguntaUsuario;
cin >> preguntaUsuario;

cout << "Si pensaras en un " << objetoUsuario << ", cual sería la respuesta a esta
pregunta?:" << endl;

char input = getYesOrNo(); //Devuelve s o n

//Distingue si es s o n para realizar los enlaces correctos a los nuevos nodos
if (input == 's')   {
        nodoNo = new ABNodo(mensaje);
        nodoSi = new ABNodo (objetoUsuario);
} else {
        nodoSi = new ABNodo (mensaje);
        nodoNo = new ABNodo (objetoUsuario);
}

cout << "Gracias! Has expandido mi conocimiento!" << endl;

//La pregunta del usuario es ahora el mensaje del nodo padre actual
mensaje = preguntaUsuario;
```

BIBLIOGRAFÍA

Sedgewick, R. *Algorithms in C++, Parts 1–4: Fundamentals, Data Structure, Sorting, Searching*. Third Edition, Addison Wesley Professional, ISBN-13: 978-0-201-35088-3

C++ con Clase [en línea] http://c.conclase.net/

OFlaherty, K. “20 Questions Guessing Game using Binary Trees” [en línea] [C# Corner] http://www.c-sharpcorner.com/UploadFile/4a950c/5079/