Universidad Autónoma de Baja California

Ingeniero en Software y Tecnologías Emergentes

Estructura de Datos 2022-2

Unidad 4:

Arboles Binarios

Nombre	Luis Eduardo Galindo Amaya
Matricula	1274895
Grupo	341
Docente	Itzel Barriba Cázares

Arboles

Objetivo

Aplicar el uso de estructuras de datos para implementar listas enlazadas dobles para la resolución de problemas reales en el área de ingeniería, con honestidad y responsabilidad

Equipo y material de apoyo:

- Computadora Personal (PC)
- Ambiente de desarrollo o IDE (CodeBlocks, Geany, VSCODE)

Fundamentos:

Un árbol es una estructuras de datos similar a una lista enlazada, pero en lugar de que cada nodo apunte simplemente al siguiente nodo de forma lineal, cada nodo apunta a una serie de nodos.

Una árbol es un ejemplo de una estructuras de datos no-lineal. Una estructura de árbol es una manera de representar la naturaleza jerárquica de una estructura en forma gráfica.

En los árboles (ADT), el orden de los elementos no es importante. Si necesitamos información de pedido, se pueden usar estructuras de datos lineales como listas enlazadas, pilas, colas, etc.

Recorrido PreOrden

El recorrido PreOrden es definido de la siguiente manera:

- Visita la raíz
- Atraviesa el subárbol izquierdo en PreOrden
- Atraviesa el subárbol derecho en PreOrden

Recorrido InOrden

En el recorrido InOrden la raíz es visitada entre los subárboles.

- Atraviesa el subárbol izquierdo en InOrden
- Visita la raíz
- Atraviesa el subárbol derecho en InOrden

Recorrido PostOrden

En el recorrido PostOrden la raíz es visitada después de ambos subárboles. El recorrido PostOrden es
definido de la siguiente manera:

- Atraviesa el subárbol izquierdo en PostOrden
- Atraviesa el subárbol derecho en PostOrden
- Visita la raíz

Recorrido Level orden

En el recorrido LevelOrden es definido de la siguiente manera:

- Visita la raíz
- Mientras se atraviesa el nivel, (mantenga todos los elementos en el nivel (+1 en la cola
- Ir al siguiente nivel y visitar todos los nodos de ese nivel
- Repita esto hasta completar todos los niveles

Desarrollo de la practica:

Parte 1: Implementar un programa para generar un árbol binario y los 4 diferentes tipos de recorridos, PreOrden, InOrden, PostOrden, LevelOrder basarse en las funciones definidas en la presentación deberas utilizar los programas antes hechos de pilas.

Parte 2: Implementar un programa para generar un árbol binario y los 4 diferentes tipos

de recorridos, PreOrden, InOrden, PostOrden, basarse en las funciones recursivas definidas en la presentación deberás utilizar los programas antes hechos de pilas.

Explicación del funcionamiento

El árbol que se crea con el método insertNode funciona de la siguiente manera:

1. Se pasa como parámetro una dirección a un árbol y un valor
2. Si el árbol esta vacío se toma el valor como la raíz del árbol
3. Cuando el árbol tiene una raíz se evalúan la condiciones
   - **Si** es menor a al valor del nodo el valor se vuelve a evaluar en la dirección del nodo izquierdo
   - **Si** es mayor a al valor del nodo el valor se vuelve a evaluar en la dirección del nodo derecho
   - **Si** es igual el nodo se descarta por completo y se pasa al siguiente nodo

Cada una de las funciones de ordenamiento requiere, que se le pase una dirección de memoria correspondiente al árbol al que se le desea aplicar, para cada uno de los posibles algoritmos hay dos soluciones, excepto levelorder, una recursiva y una lineal, las funciones lineales por lo general necesitan de otra estructura de datos para realizar el ordenamiento adecuadamente. Por otro lado las funciones recursivas no requieren de otro tipo de estructuras de datos para realizar los procedimientos.

Al usar el **preorder** lo que estamos haciendo es, mostrar el valor, recorrer el árbol por la izquierda y agregando las direcciones de los nodos en el stack hasta llegar a un nodo vacio una vez allí se termina y procedemos a vaciar los valores en el stack por la derecha hasta que este vació,

en el **Inorder** es bastante similar, la principal diferencia es el momento cuando mostramos el valor, que es inmediatamente después de sacarlo del stack.

Postorder recorre el árbol por la derecha y coloca los valores en un stack hasta que el nodo sea nulo, una vez llegado a ese punto se revisa el valor del primer elemento y si es nulo y la pila no esta vaciá se revisa si el primer elemento de la lista, si es diferente

al valor anterior y es diferente a nulo se elimina del stack se imprime el valor y se pasa por el nodo derecho.

Levelorder se utiliza una cola, el primer nodo de la cola es la raíz, si la cola no esta vaciá se continua con la iteración, se imprime el valor y se evaluá las siguientes condiciones :

- si el nodo izquierdo no esta vació se agrega a la cola
- si el nodo derecho no esta vació se agrega la cola

Conclusiones

A partir de esta practica puede entender como es posible guardar un árbol en memoria y como podríamos almacenarlo dentro de una archivo, algunos códigos fueron un poco complicados de implementar debido a que requerían que usáramos practicas previas para la implementación de los algoritmos, pero solo basto con un par de cambios en una cuantas lineas de código para poder usarlas sin ningún problema .

Fuentes

Joyanes Aguilar, L. (2008). *Fundamentos de programación: Algoritmos, estructura de datos y objetos* (4a. ed. --.). Madrid: McGraw-Hill.

Ejecución

Al entrar al programa se encontrara con el siguiente menú, el cual le mostrara todas las posibles, ingrese el numero de opción para probar cada opción

```
----------------------------
       Menu principal
----------------------------
Misc.
 1. Autogenerar y probar
 2. Level order

No recursivos
 3. Preorder no recursivo
 4. Postorder no recursivo
 5. Inorder no recursivo

Recursivos
 6. Preorder recursivo
 7. Postorder recursivo
 8. Inorder recursivo
----------------------------
 0. Salir

op>
```

Figura 1: Menú principal

1. Auto generar y probar

genera una lista de nodos aleatorios y prueba todos los métodos, si al auto generar un nodo se repite aparece la leyenda “dup” al lado del numero:

```
numero de nodos: 13

The numbers being placed in the tree are:
 11 9 7 11:dup 10 11:dup 6 1 5 11:dup 4 8 9:dup

LevelOrder
 11 9 7 10 6 8 1 5 4

preOrder
 Recursivo:    11 9 7 6 1 5 4 8 10
 No Recursivo: 11 9 7 6 1 5 4 8 10

inOrder
 Recursivo:    1 4 5 6 7 8 9 10 11
 No Recursivo: 1 4 5 6 7 8 9 10 11

postOrder
 Recursivo:    4 5 1 6 8 7 10 9 11
 No Recursivo: 4 5 1 6 8 7 10 9 11
```

Figura 2: Autogenera los nodos y les aplica los algoritmos

2. Level order

Captura los nodos y muestra el level order de los nodos, si un al capturar un nodo se repite el nodo aparece la leyenda “dup” y se ignore:

```
numero de nodos: 5
nodo 1: 1
nodo 2: 4
nodo 3: 5
nodo 4: 5
:dupnodo 5: 8

Level Order:
1 4 5 8
```

Figura 3: Level order

3. Preorder no recursivo

Captura los nodos y muestra el Preorder de los nodos, si un al capturar un nodo se repite el nodo aparece la leyenda “dup” y se ignore:

```
numero de nodos: 5
nodo 1: 1
nodo 2: 2
nodo 3: 3
nodo 4: 4
nodo 5: 6

PreOrder no recursivo:
1 2 3 4 6
```

Figura 4: Preorder no recursivo

4. Postorder no recursivo

Captura los nodos y muestra el Postorder de los nodos, si un al capturar un nodo se repite el nodo aparece la leyenda “dup” y se ignore:

```
numero de nodos: 4
nodo 1: 1
nodo 2: 5
nodo 3: 3
nodo 4: 7

PostOrder no recursivo
3 7 5 1
```

Figura 5: Postorder no recursivo

5. Inorder no recursivo

Captura los nodos y muestra el Inorder de los nodos, si un al capturar un nodo se repite el nodo aparece la leyenda “dup” y se ignore:

```
numero de nodos: 4
nodo 1: 5
nodo 2: 6
nodo 3: 2
nodo 4: 9

InOrder no recursivo
2 5 6 9
```

Figura 6: Inorder no recursivo

6. PreOrder recursivo

Captura los nodos y muestra el PreOrder de los nodos, si un al capturar un nodo se repite el nodo aparece la leyenda “dup” y se ignore:

```
numero de nodos: 2
nodo 1: 5
nodo 2: 7

PreOrder recursivo
5 7
```

Figura 7: PreOrder recursivo

7. PosOrder recursivo

Captura los nodos y muestra el PosOrder de los nodos, si un al capturar un nodo se repite el nodo aparece la leyenda “dup” y se ignore:

```
numero de nodos: 4
nodo 1: 1
nodo 2: 2
nodo 3: 7
nodo 4: 8

PostOrder recursivo
8 7 2 1
```

Figura 8: PosOrder recursivo

8. Inorder recursivo

Captura los nodos y muestra el Inorder de los nodos, si un al capturar un nodo se repite el nodo aparece la leyenda “dup” y se ignore:

```
numero de nodos: 5
nodo 1: 1
nodo 2: 4
nodo 3: 9
nodo 4: 4
:dupnodo 5: 3

InOrder recursivo
1 3 4 9
```

Figura 9: Inorder recursivo

Análisis de Resultados

En esta practica se utilizo todo lo aprendido hasta ahora, desde listas enlazadas, pilas, listas, colas, recursion y memoria dinámica, se logro resolver todos los problemas sin tener que recurrir a fuentes externas. Sin embargo en algunos de los métodos de esta usando memoria dinámica y no se esta liberando adecuadamente:

- InsertNode
- PostOrderNonRecursive

Código main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "./lib/binaryTreeStruct.h"
#include "./lib/binaryTreeNonRecursive.h"
#include "./lib/binaryTreeRecursive.h"

void insertNode(TreeNodePtr *treePtr, int value);
TreeNodePtr captureNodes(int ntest);
void autogenerateTree(int ntest);

int main() {
  TreeNodePtr rootPtr;
  int op, ntest;

  while (1) {
    printf("\n"
           "----------------------------\n"
           "       Menu principal \n"
           "----------------------------\n"
           "Misc.\n"
           " 1. Autogenerar y probar \n"
           " 2. Level order \n"
           "\n"
           "No recursivos\n"
           " 3. Preorder no recursivo \n"
           " 4. Postorder no recursivo \n"
           " 5. Inorder no recursivo \n"
           "\n"
           "Recursivos\n"
           " 6. Preorder recursivo \n"
           " 7. Postorder recursivo \n"
           " 8. Inorder recursivo \n"
           "----------------------------\n"
           " 0. Salir\n"
           "\n"
           "op> ");

    scanf("%d", &op);

    if(op == 0) {               /* salir */
      exit(EXIT_SUCCESS);
    }

    printf("\nnumero de nodos: ");
    scanf("%d", &ntest);

    if(op == 1) {               /* pruebas */
```

```
      autogenerateTree(ntest);
      continue;
    }

    rootPtr = captureNodes(ntest);

    switch (op) {

    case 2:                     /* LevelOrder */
      printf("Level Order:\n");
      LevelOrder(rootPtr);
      break;

    case 3:                     /* PreOrderNonRecursive */
      printf("PreOrder no recursivo: \n");
      PreOrderNonRecursive(rootPtr);
      break;

    case 4:                     /* PostOrderNonRecursive */
      printf("PostOrder no recursivo\n");
      PostOrderNonRecursive(rootPtr);
      break;

    case 5:                     /* InOrderNonRecursive */
      printf("InOrder no recursivo\n");
      InOrderNonRecursive(rootPtr);
      break;

    case 6:                     /* PreOrder */
      printf("PreOrder recursivo\n");
      PreOrder(rootPtr);
      break;

    case 7:                     /* PostOrder */
      printf("PostOrder recursivo\n");
      PostOrder(rootPtr);
      break;

    case 8:                     /* InOrder */
      printf("InOrder recursivo\n");
      InOrder(rootPtr);
      break;
    }

    printf("\n");
  }


  return 0;
}
```

```
TreeNodePtr captureNodes(int ntest){
  TreeNodePtr treePtr = NULL;
  int op;

  for (int i = 1; i < ntest+1; i++) {
    printf("nodo %d: ",i);
    scanf("%d",&op);
    insertNode(&treePtr, op);
  }

  puts("");
  return treePtr;
}

void autogenerateTree(int ntest) {
  TreeNodePtr rootPtr = NULL;
  srand(time(NULL));
  puts("\n"
       "The numbers being placed in the tree are:");

  /* genera datos aleatorios */
  for (int i = 0; i < ntest; i++) {
    int item = rand() % 15;
    printf(" %d", item);
    insertNode(&rootPtr, item);
  }

  printf("\n\nLevelOrder\n ");
  LevelOrder(rootPtr);

  printf("\n\n"
         "preOrder\n"
         " Recursivo:    ");
  PreOrder(rootPtr);
  printf("\n"
         " No Recursivo: ");
  PreOrderNonRecursive(rootPtr);

  printf("\n\n"
         "inOrder \n"
         " Recursivo:    ");
  InOrder(rootPtr);
  printf("\n"
         " No Recursivo: ");
  InOrderNonRecursive(rootPtr);

  printf("\n\n"
         "postOrder\n"
         " Recursivo:    ");
  PostOrder(rootPtr);
  printf("\n"
```

```
          " No Recursivo: ");
  PostOrderNonRecursive(rootPtr);

  printf("\n");
}

void insertNode(TreeNodePtr *treePtr, int value) {
  if (*treePtr) { /* si el nodo tiene contenido (no es NULL) */

    if (value < (*treePtr)->data) {
      /* si el valor es menor al valor del nodo actual inserta
         a la izquierda */
      insertNode(&((*treePtr)->left), value);
      return;
    }

    if (value > (*treePtr)->data) {
      /* si el valor es mayor al valor del nodo actua inserta
         a la izquierda */
      insertNode(&((*treePtr)->right), value);
      return;
    }

    /* no inserta el valor si es igual */
    printf(":dup");
    return;
  }

  if ((*treePtr = malloc(sizeof(BinaryTreeNode)))) {
    /* si el nodo esta vacio (es NULL) crea el nuevo nodo */
    (*treePtr)->data = value;
    (*treePtr)->left = NULL;
    (*treePtr)->right = NULL;
    return;
  }

  /* si no se pudo reveservar memoria se muestra el siguiente
     mensaje */
  printf("%d not inserted. No memory available.\n", value);
  exit(EXIT_FAILURE);
}
```

Código en binaryTreeRecursive.h

```c
void PreOrder(BinaryTree *root) {
  if (!root)
    return;                    /* termina si la raiz es nula */

  printf("%d ", root->data);
  PreOrder(root->left);
  PreOrder(root->right);
}

void InOrder(BinaryTree *root) {
  if (!root)
    return;                    /* termina si la raiz es nula */

  InOrder(root->left);
  printf("%d ", root->data);
  InOrder(root->right);
}

void PostOrder(BinaryTree *root) {
  if (!root)
    return;                    /* termina si la raiz es nula */

  PostOrder(root->left);
  PostOrder(root->right);
  printf("%d ", root->data);
}
```

Código en binaryTreeNonRecursive.h

```
#include "queueA.h"
#include "stackA.h"

void PreOrderNonRecursive(BinaryTree *root) {
  Stack s;
  stackInit(&s);

  while (1) {
    while (root) {
      printf("%d ", root->data); /* mostrar el valor del nodo */
      stackPush(&s, root);/* agregar al stack */
      root = root->left;
    }

    if (stackEmpty(&s))/* terminar loop infinito */
      break;

    root = stackPop(&s);
    root = root->right; /* moverse a la derecha */
  }
  stackFree(&s);
}

void InOrderNonRecursive(BinaryTree *root) {
  Stack s;
  stackInit(&s);

  while (1) {
    while (root) {
      stackPush(&s, root);/* agregar a al pila */
      root = root->left; /* moverse a la izquierda */
    }

    if (stackEmpty(&s))
      break; /* terminar el loop */

    root = stackPop(&s);
    printf("%d ", root->data);
    root = root->right; /* moverse a al edrecha si esta vacio */
  }

  stackFree(&s);
}

void PostOrderNonRecursive(BinaryTree *root) {
  Stack s;
  stackInit(&s);
  BinaryTree *previous = NULL;
```

```
  do {
    while (root ≠ NULL) { /* continua mierntras raiz no esta esta vacio */
      stackPush(&s, root);
      root = root→left; /* moverse a la izquierda*/
    }

    while (root = NULL && !stackEmpty(&s)) {
      root = stackTop(&s);
      /* continuar mientras sea igual al previo*/
      if (root→right = NULL || root→right = previous) {
        printf("%d ", root→data);
        stackPop(&s);
        previous = root;
        root = NULL;
      } else {
        root = root→right;
      }
    }
  } while (!stackEmpty(&s));
}

void LevelOrder(BinaryTree *root) {
  BinaryTreeNode *temp;
  Queue q;
  queueInit(&q);

  if (!root) /* si no hay nodos terminar */
    return;

  enqueue(&q, root); /* agregar el primer nodo a la cola para iniciar la
                        iteracion  */

  while (!queueEmpty(&q)) {
    temp = dequeue(&q);
    printf("%d ", temp→data);

    if (temp→left)
      enqueue(&q, temp→left); /* agregar primero el nodo a al quierda */

    if (temp→right)
      enqueue(&q, temp→right); /* despeus a la derecha */
  }

  queueFree(&q);
}
```