جامعة المنارة
كلية الهندسة
قسم الهندسة المعلوماتية

مقرر الخوارزميات و بنى المعطيات ١

جلسة العملي السادسة

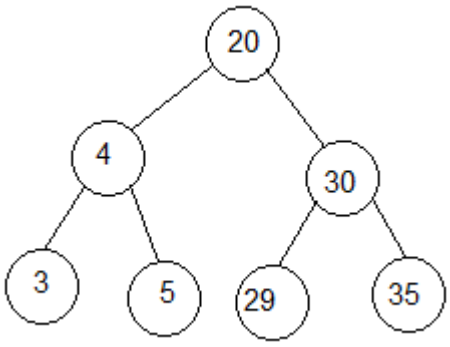
(الفصل الثاني ٢٠٢٣-٢٠٢٤)

الأشجار الثنائية

تمرين ١:

اكتب برنامجا بلغة C++ يقوم بما يلي:

- ببناء الشجرة الموضحة في الشكل أدناه
- التجول في الشجرة بالأساليب الثلاثة : preorder inorder postorder
- حذف العقدة رقم 4
- اختبار وجود عقدة ضمن الشجرة تحمل الرقم 13
- اختبار وجود عقدة ضمن الشجرة تحمل الرقم 29



الحل:

```
#include <iostream>
using namespace std;
```

تعريف الصف الممثل لعقد الشجرة:

```
class BSTnode{
 public:
 int data;
```

```
 BSTnode *left, *right;

 BSTnode(){
  left=right=NULL;
 }

 BSTnode (int dat, BSTnode *lft=NULL, BSTnode
*rgt=NULL){
  data = dat;
  left = lft;
  right = rgt;
 }
};
```

تعريف الصف الممثل للشجرة الثنائية:

```
class BST{
 public:
  BSTnode *root;

  BST(){
   root = NULL;
  }
```

تابع اختبار الشجرة في حال كانت فارغة:

```
  bool isEmpty() { return root==NULL; }
```

تابع إضافة عقدة للشجرة:

```
  void insert (int el, BSTnode* &myroot){
   if (myroot==NULL){
    myroot = new BSTnode (el,0,0);
```

```
 cout << el << " inserted\n";
 } else if (el<myroot->data){
  insert (el,myroot->left);
 } else if (el>myroot->data){
  insert (el,myroot->right);
 }
}
```

تابع اختبار انتماء عقدة للشجرة:

```
bool search(int el, BSTnode* &start){
 if (start!=NULL){
  if (el<start->data){
   return search(el,start->left);
  } else if (el>start->data){
   return search (el,start->right);
  } else {
   return true;
  }
 }
 return false;
}
```

تابع مسح عقد الشجرة بشكل inorder:

```
void inorder(BSTnode* &myroot){
 if (myroot->left!=NULL)
  inorder(myroot->left);
 if (myroot!=NULL){      visit (myroot);    }
 if  (myroot->right!=NULL){       inorder(myroot->right);
}
 }
```

تابع مسح عقد الشجرة بشكل preorder:

```
 void preorder(BSTnode* &myroot){
  if (myroot!=NULL){     visit (myroot);    }
  if (myroot->left!=NULL)     preorder(myroot->left);
  if (myroot->right!=NULL){     preorder(myroot->right);
}
  }
```

تابع مسح عقد الشجرة بشكل postorder:

```
 void postorder(BSTnode* &myroot){
 if (myroot->left!=NULL)     postorder(myroot->left);
  if   (myroot->right!=NULL){           postorder(myroot-
>right);   }
   if (myroot!=NULL){      visit (myroot);    }
 }
```

تابع لزيارة عقدة من عقد الشجرة:

```
 void visit(BSTnode* &node){ cout << node->data<<"-
>"; }
```

تابع حذف عقدة من عقد الشجرة:

```
 void      delete_node      (BSTnode**      node){          if
(node==NULL) return;
  BSTnode *old_node = *node;
  if ((*node)->left == NULL){
   *node = (*node)->right;
   delete old_node;
```

```
 } else if ((*node)->right == NULL){
  *node = (*node)->left;
  delete old_node;
 } else {
  /*    NODE with 2 children   */
  // find inorder predecessor
  BSTnode **pred = &(*node)->left;
  while ((*pred)->right != NULL) {
   *pred = (*pred)->right;
  }
  // swap pred's data
  swap ((*pred)->data, (*node)->data);
  delete_node (pred);
 }
}
```

تابع البحث عن موقع عقدة ضمن الشجرة:

```
BSTnode** find_node (int data){
 BSTnode** node = &root;
   while (*node != NULL) {
       if (data < (*node)->data)
         node = &(*node)->left;
       else if ((*node)->data < data)
          node = &(*node)->right;
       else
          break;
   }
   return node;
 }
};
```

التابع الرئيسي:

```
int main(){
 BST b;
 cout<<"isEmpty "<<b.isEmpty()<<endl;
 b.insert (20,b.root);
 cout<<"isEmpty "<<b.isEmpty()<<endl;
 b.insert (4,b.root);
 b.insert (30,b.root);
 b.insert (3,b.root);
 b.insert (5,b.root);
 b.insert (29,b.root);
 b.insert (35,b.root);

 cout << "inorder traversal - ";
 b.inorder (b.root);
 cout << endl;

 cout << "preorder traversal - ";
 b.preorder (b.root);
 cout << endl;

 cout << "postorder traversal - ";
 b.postorder (b.root);
 cout << endl;

 cout << "deleting 4\n";
 b.delete_node (b.find_node(4));

 if (b.search(13,b.root))
  cout << "13 found!\n";
 else
  cout << "13 not found\n";
```

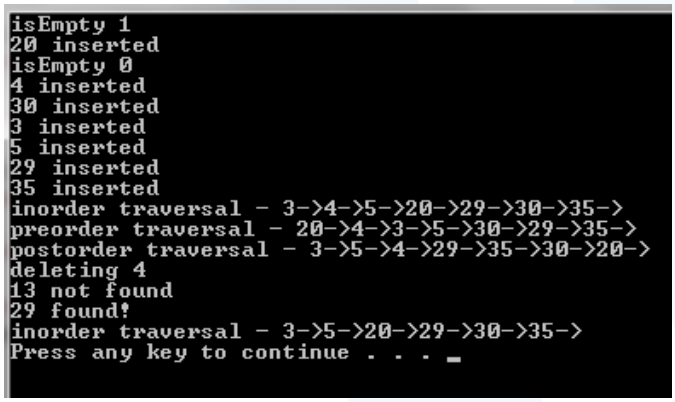
```
if (b.search(29,b.root))
 cout << "29 found!\n";
else
 cout << "29 not found\n";

cout << "inorder traversal - ";
b.inorder (b.root);
cout << endl;

 system ("pause");

return 0;
}
```

فيكون الخرج الناتج على الشكل التالي:



تمرين ٢:

يبين البرنامج التالي طريقة أخرى لبرمجة مسح عقد الشجرة بالطرق الثلاثة المذكورة سابقاً

```
#include <iostream>
using namespace std;

    class node
      {
              public:  int data;
        node *left;
        node *right;
      };

       node *tree=NULL;
       node *insert(node *tree,int ele);

       void preorder(node *tree);
       void inorder(node *tree);
       void postorder(node *tree);
       int count=1;

       void main()
       {
          int ch,ele;
          do
          {
             cout<<"\n\t1----INSERT A NODE IN A
BINARY TREE.";
```

```
            cout<<"\n\t2----PRE-ORDER
TRAVERSAL.";
            cout<<"\n\t3----IN-ORDER TRAVERSAL.";
            cout<<"\n\t4----POST-ORDER
TRAVERSAL.";
            cout<<"\n\t5----EXIT.";
            cout<<"\n\tENTER CHOICE::";
            cin>>ch;
            switch(ch)
            {
                case 1:
                cout<<"\n\tENTER THE ELEMENT::";
                cin>>ele;
                tree=insert(tree,ele);
                break;

                case 2:
                cout<<"\n\t****PRE-ORDER
TRAVERSAL OF A TREE****  ";
                preorder(tree);
                break;

                case 3:
                cout<<"\n\t****IN-ORDER
TRAVERSAL OF A TREE****  ";
                inorder(tree);
                break;

                case 4:
                cout<<"\n\t****POST-ORDER
TRAVERSAL OF A TREE****  ";
```

```
                postorder(tree);
                break;

                case 5:
                exit(0);
            }
        }while(ch!=5);
    }

    node *insert(node *tree,int ele)
    {
        if(tree==NULL)
        {
            tree=new node;
            tree->left=tree->right=NULL;
            tree->data=ele;
            count++;
        }
        else
              if(tree->data > ele)
        tree->left=insert(tree->left,ele);
        else if (tree->data < ele)
                              tree->right=insert(tree-
>right,ele);
                        return(tree);
    }

    void preorder(node *tree)
    {
        if(tree!=NULL)
        {
```

```
        cout<<tree->data<<"  ";
        preorder(tree->left);
        preorder(tree->right);
    }
}

void inorder(node *tree)
{
    if(tree!=NULL)
    {
        inorder(tree->left);
        cout<<tree->data<<"    ";
        inorder(tree->right);
    }
}

void postorder(node *tree)
{
    if(tree!=NULL)
    {
        postorder(tree->left);
        postorder(tree->right);
        cout<<tree->data<<"   ";
    }
}
```