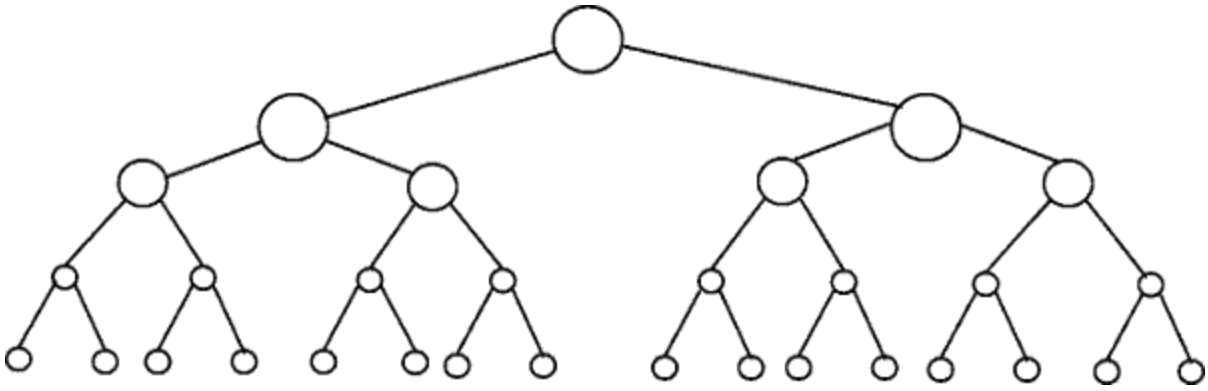
Assignment 4

Binary Search Trees

Name: Youssef Mohamed Aly Kishk
ID: 3812

Name: Nouran Youssef Ibrahim
ID: 4960

Name: Aghapy Gad saiid
ID: 4821



Introduction:

We extend the concept of linked data structures to structure containing nodes with more than one self-referenced field. A binary tree is made of nodes, where each node contains a "left" reference, a "right" reference, and a data element. The topmost node in the tree is called the root.

Every node (excluding a root) in a tree is connected by a directed edge from exactly one other node. This node is called a parent. On the other hand, each node can be connected to arbitrary number of nodes, called children. Nodes with no children are called leaves, or external nodes. Nodes which are not leaves are called internal nodes. Nodes with the same parent are called siblings.

Problem statement:

In this assignment you are required to implement a binary search tree based spell checking system. Initially you will be given a dictionary of words to build your binary search tree; the comparison between strings will be based on strcmp function available in c function. During building your BST, you are required to ensure that your tree is balanced. After finishing the BST you are required to print the height of the generated tree. Then you will be provided with a sentence to spell check. Then you will be required to determine if the word is in your tree or not
1. If it is in your tree then you will print that the word is correct
2. If it isn't then you will print three suggestions for the correct word, the word in the leaf node you reached, the word in the predecessor node and the node in the successor node.

Program Algorithm

1. First we read the 3000 words dictionary file into an array of structs, where each struct contain only an array of characters.

```
typedef struct fileWords{
    char word[20];
}fileWords;
fileWords words[3000];
```

   - Here are the used function to read the words from the file

```
void dealWithInputFile(){
    int count =0;
    FILE *af = fopen("English Dictionary.txt","r");
    char line[50];
    while(!feof(af)){
        fgets(line,50,af);
        strtok(line,"\n");
        strcpy(words[count].word,line);
        count++;
    }
}
```

2. We hold the file size in a variable n

```
int n = sizeof(words)/sizeof(words[0]);
```

3. We start building a balanced binary search tree from the array of structs of the file words we have now.
   - Function we used to build balanced BST

```
node * sortedArrayToBST(fileWords arr[], int start, int end){
    if (start > end)
      return NULL;

    //Get the middle element and make it root
    int mid = (start + end)/2;
    node *root = newNode(arr[mid].word);

    // construct the left subtree
    root->left =  sortedArrayToBST(arr, start, mid-1);

    // construct the right subtree
    root->right = sortedArrayToBST(arr, mid+1, end);

    return root;
}
```

   - Here are the function to check if its balanced

```
int isBalanced(node *root){
    int l;
    int r;
    if(root == NULL)
     return 1;
    l = height(root->left);
    r = height(root->right);
    if( abs(l-r) <= 1 &&
        isBalanced(root->left) &&
        isBalanced(root->right))
      return 1;
    return 0;
}
```

4. Now we have a balanced BST holding the dictionary words, then we determine its height.
   - ➢ Determining height function

```
int height(node *r){
    if(r)
         return 1 + max(height(r->left), height(r->right));
    else
        return 0;
}
int max(int a, int b){
  return (a >= b)? a: b;
}
```

5. Now the program is ready to accept any word from the user and search for it in the BST to determine its presence or start finding the **successor**, **predecessor** and **leaf** nodes.
   - In case the word was found we print to the user that its correct
   - In case it's not found we get the successor, predecessor, leaf of this word then printed to user.
   - ➢ Used function to search for certain word in BST

```
int searching(node *root,char *word)
{
     while (root != NULL)
     {
          if (strcmp(word,root->word)<0){
               leafParentOfParent=leafParent;
               leafParent=leaf;
               leaf =root;
               root = root->left;
          }

          else if (strcmp(word,root->word)>0){
               leafParentOfParent=leafParent;
               leafParent=leaf;
               leaf =root;
               root = root->right;
          }
          else
               return 1;// if the key is found
     }
     return 0;
}
```

➢ Used function for finding successor

```
node * successor(node* suc){
    if(suc->right){
        suc = suc->right;
        while(suc->left){
            suc = suc->left;
        }
    }
    else{
        if(leafParent->right!=NULL&&strcmp(suc->word,leafParent->right->word)==0)
              suc=leafParentOfParent;
        else if(leafParent->left!=NULL&&strcmp(suc->word,leafParent->left->word)==0)
               suc=leafParent;
    }
        return suc;
}
```

➢ Used function for finding predecessor

```
node * predecessor(node *pred){
    if(pred->left){
        pred = pred->left;
        while(pred->right){
            pred = pred->right;
        }
    }
    else{
        if(leafParent->left!=NULL&&strcmp(pred->word,leafParent->left->word)==0)
               pred=leafParentOfParent;
         else if(leafParent->right!=NULL&&strcmp(pred->word,leafParent->right->word)==0)
                pred=leafParent;
    }
        return pred;
}
```

Extra Notes

- Dealing with user's input.

```
char testWord[50];
int spaces=0;
int i;
printf("Enter a word to search for: ");
scanf("%s",&testWord);
switch(searching(root,testWord)){
    case 1:printf("The word <%s> is correct\n",testWord);break;
    case 0: printf("\nWord <%s> is Wrong\n",testWord);
            printf("Leaf word is <%s>\n",leaf->word);
            printf("Successor is <%s>\n",successor(leaf)->word);
            printf("Predecessor is <%s>\n",predecessor(leaf)->word);
            break;
}
```

- BST structure and adding new node to it.

```
typedef struct node
{
    char word[50];
    struct node *left, *right;
}node;
node * newNode(char *word){
    node* n = (node*)malloc(sizeof(node));
    strcpy(n->word,word);
    n->left=NULL;
    n->right=NULL;
    return n;
}
```

Test samples

- Waiting for user to enter a word to search for.

```
The tree is balanced
Height of the tree = 12

Enter a word to search for:
```

- Correct words test.

```
The tree is balanced
Height of the tree = 12

Enter a word to search for: play
The word <play> is correct

Process returned 0 (0x0)   execution time : 1.231 s
Press any key to continue.
```

```
The tree is balanced
Height of the tree = 12

Enter a word to search for: sing
The word <sing> is correct

Process returned 0 (0x0)   execution time : 1.428 s
Press any key to continue.
```

- Un correct words test

```
The tree is balanced
Height of the tree = 12

Enter a word to search for: genious

Word <genious> is Wrong
Leaf word is <gentleman>
Successor is <generation>
Predecessor is <genetic>

Process returned 0 (0x0)   execution time : 3.854 s
Press any key to continue.
```

```
The tree is balanced
Height of the tree = 12

Enter a word to search for: studies

Word <studies> is Wrong
Leaf word is <studio>
Successor is <study>
Predecessor is <student>

Process returned 0 (0x0)   execution time : 7.758 s
Press any key to continue.
```