

DSA ASSIGNMENT 3

Name: Varun Vivek Pai

USN: 01FE23BEC126

Roll No.: 332

Div: C

1. You are given a set of integers: [45, 23, 78, 12, 34, 56, 89, 67, 05, 99]. Construct a Binary Search Tree (BST) by inserting these elements in the given order. Subsequently, demonstrate the step-by-step process for:

- a) Deleting the node with value 34 (a leaf node).
- b) Deleting the node with value 78 (a node with two children).
- c) Inserting the value 40 into the modified BST. Draw the state of the BST after each operation.

Soln:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
typedef struct Node
```

```
{
```

```
    int data;
```

```
    struct Node* left;
```

```
    struct Node* right;
```

```
} Node;
```

```
Node* insert(Node* root, int data)
```

```
{
```

```
    Node* nw, *cur, *parent;
```

```
    nw = (Node*) malloc(sizeof(Node));
```

```
    if (nw)
```

```
    {
```

```
        nw->data = data;
```

```

nw->left=nw->right=NULL;
if (root == NULL)
    return nw;
for(cur = root, parent = NULL; cur != NULL;)
{
    parent = cur;
    if (nw->data < cur->data)
        cur = cur->left;
    else
        cur = cur->right;
}
if (nw->data < parent->data)
    parent->left = nw;
else
    parent->right = nw;
}
else
{
    printf("\nNode allocation failed\n");
}
return root;
}

```

```

Node* deleteNode (Node*root, int data)
{
    struct Node *par, *cur, *temp;
    if (root == NULL)
    {
        printf ("\nEmpty Tree\n");
    }
}

```

```

    return root;
}
par = NULL;
cur = root;
while (cur != NULL && cur->data != data)
{
    par = cur;
    if (data < cur->data)
        cur = cur->left;
    else
        cur = cur->right;
}
if (cur == NULL)
{
    printf ("\nNode not found\n");
    return root;
}
if (cur->left != NULL && cur->right != NULL)
{
    temp = par = cur;
    cur = cur->left;
    while (cur->right != NULL)
    {
        par = cur;
        cur = cur->right;
    }
    temp->data = cur->data;
}
if (cur->left != NULL && cur->right == NULL)

```

```

{
    if (cur == root)
        root = root -> left;
    else
    {
        if (cur == par -> left)
            par -> left = cur -> left;
        else
            par -> right = cur -> left;
    }
}
else if (cur -> left == NULL && cur -> right != NULL)
{
    if (cur == root)
        root = root -> right;
    else
    {
        if (cur == par -> left)
            par -> left = cur -> right;
        else
            par -> right = cur -> right;
    }
}
else
{
    if (cur == root)
        root = NULL;
    else
    {

```

```

        if (par -> left == cur)
            par -> left = NULL;
        else
            par -> right = NULL;
    }
}
free(cur);
return root;
}

```

```

void inorder(Node* root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

```

```

int main()
{
    int values[10] = {45, 23, 78, 12, 34, 56, 89, 67, 5, 99}, n=10, i;

    Node* root = NULL;
    for (i = 0; i < n; i++)
        root = insert(root, values[i]);

    printf("Inorder after inserting all elements:\n");
}

```

```

inorder(root);

root = deleteNode(root, 34);

printf("\n\nInorder after deleting 34 (leaf node):\n");

inorder(root);

root = deleteNode(root, 78);

printf("\n\nInorder after deleting 78:\n");

inorder(root);

root = insert(root, 40);

printf("\n\nInorder after inserting 40:\n");

inorder(root);

printf("\n");

return 0;

}

```

The screenshot displays a C++ IDE with a project named 'bst'. The main.c file contains the following code:

```

122 }
123
124
125 int main()
126 {
127     int values[10] = {45, 23, 78, 15, 12, 34, 45, 56, 67, 89, 99};
128     Node* root = NULL;
129     for (i = 0; i < n; i++)
130         root = insert(root, values[i]);
131
132     printf("Inorder after inserting all elements:\n");
133     inorder(root);
134
135     root = deleteNode(root, 34);
136     printf("Inorder after deleting 34 (leaf node):\n");
137     inorder(root);
138
139     root = deleteNode(root, 78);
140     printf("Inorder after deleting 78:\n");
141     inorder(root);
142
143     root = insert(root, 40);
144     printf("Inorder after inserting 40:\n");
145     inorder(root);
146     printf("\n");
147     return 0;
148 }
149
150

```

The console output shows the following sequence of operations and their results:

```

Inorder after inserting all elements:
5 12 23 34 45 56 67 78 89 99
Inorder after deleting 34 (leaf node):
5 12 23 45 56 67 78 89 99
Inorder after deleting 78:
5 12 23 45 56 67 89 99
Inorder after inserting 40:
5 12 23 40 45 56 67 89 99
Process returned 0 (0x0)   execution time : 0.046 s
Press any key to continue.

```

The IDE interface includes a file explorer on the left showing the project structure, a code editor in the center, and a console window on the right. The status bar at the bottom indicates the current file is 'main.c' and the cursor is at line 141, column 39, position 3131.

2. Consider an initially empty AVL tree. Insert the following sequence of values into the AVL tree, one by one: [10, 20, 30, 25, 28, 05, 08]. For each insertion, clearly show the balance factor of all affected nodes and demonstrate any necessary rotations (single or double) performed to maintain the AVL tree property. Draw the tree after each insertion and subsequent balancing.

Soln:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
struct Node
```

```
{  
    struct Node *left, *right;  
    int h, key;  
};
```

```
int height (struct Node *cur)
```

```
{  
    if (cur == NULL)  
        return 0;  
    return cur -> h;  
}
```

```
struct Node * getNode (int data)
```

```
{  
    struct Node *nw;  
    nw = malloc (sizeof (struct Node));  
    nw -> key = data;  
    nw -> left = nw -> right = NULL;  
    nw -> h = 1;  
    return nw;
```

```
}
```

```
int max (int a, int b)
```

```
{
```

```
    return (a > b) ? a : b;
```

```
}
```

```
struct Node *LeftLeftRotation (struct Node *x)
```

```
{
```

```
    struct Node *y;
```

```
    y = x -> right;
```

```
    x -> right = y -> left;
```

```
    y -> left = x;
```

```
    x -> h = max(height(x -> left), height (x -> right)) + 1;
```

```
    y -> h = max(height(y -> left), height (y -> right)) + 1;
```

```
    return y;
```

```
}
```

```
struct Node *RightRightRotation (struct Node *x)
```

```
{
```

```
    struct Node *y;
```

```
    y = x -> left;
```

```
    x -> left = y -> right;
```

```
    y -> right = x;
```

```
    x -> h = max(height(x -> left), height (x -> right)) + 1;
```

```
    y -> h = max(height(y -> left), height (y -> right)) + 1;
```

```
    return y;
```

```
}
```



```

struct Node * insert (struct Node *cur, int data)
{
    int bal = 0;
    if (cur == NULL)
        return getNode (data);
    if (data < cur -> key)
        cur -> left = insert (cur -> left, data);
    else if (data > cur -> key)
        cur -> right = insert (cur -> right, data);
    else
    {
        printf ("\nNode already exists\n");
        return cur;
    }
    cur -> h = max(height(cur -> left), height (cur -> right)) + 1;
    bal = height (cur -> left) - height (cur -> right);
    if (bal > 1 && cur -> left -> key)
        return RightRightRotation (cur);
    else if (bal < -1 && data > cur -> right -> key)
        return LeftLeftRotation(cur);
    else if (bal > 1 && data < cur -> left -> key)
    {
        cur -> left = LeftLeftRotation(cur -> left);
        return RightRightRotation(cur);
    }
    else if (bal < -1 && data < cur -> right -> key)
    {
        cur -> right = RightRightRotation (cur -> right);
        return LeftLeftRotation(cur);
    }
}

```

```
    }  
    return cur;  
}
```

```
void preorder (struct Node *cur)  
{  
    if (cur == NULL)  
        return;  
    printf ("%d ", cur -> key);  
    preorder (cur -> left);  
    preorder (cur -> right);  
}
```

```
int main()  
{  
    struct Node *root = NULL;  
    int value[7] = {10, 20, 30, 25, 28, 5, 8}, i;  
    for (i = 0; i < 7; i++)  
    {  
        root = insert(root, value[i]);  
        printf("\nPreorder display of AVL Tree after inserting %d:\n", value[i]);  
        preorder(root);  
        printf("\n");  
    }  
    return 0;  
}
```

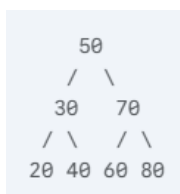
The screenshot shows a C++ IDE with a project named 'main.c [avl] - CodeBlocks 25.03'. The code implements an AVL tree with functions for insertion, rotation, and preorder traversal. The execution output displays the preorder traversal results after inserting a series of values: 10, 20, 30, 25, 28, 5, and 8. The final preorder sequence is 25 10 5 8 20 30 28. The process returned 0 (0x0) and took 0.110 seconds to execute.

```

main.c [avl] - CodeBlocks 25.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global> main.c: int
main.c x main.c x
81 cur -> right = RightRightRotation(cur);
82 return LeftLeftRotation(cur);
83 }
84 return cur;
85 }
86 }
87 void preorder(struct Node *cur)
88 {
89 if (cur == NULL)
90 return;
91 printf("%d ", cur->key);
92 preorder(cur->left);
93 preorder(cur->right);
94 }
95
96 int main()
97 {
98 struct Node *root = NULL;
99 int value[] = {10, 20, 30, 25, 28, 5, 8};
100 for (i = 0; i < 7; i++)
101 {
102 root = insert(root, value[i]);
103 preorder(root);
104 printf("\n");
105 }
106 return 0;
107 }
108
109
Preorder display of AVL Tree after inserting 10:
10
Preorder display of AVL Tree after inserting 20:
10 20
Preorder display of AVL Tree after inserting 30:
20 10 30
Preorder display of AVL Tree after inserting 25:
20 10 30 25
Preorder display of AVL Tree after inserting 28:
25 20 10 30 28
Preorder display of AVL Tree after inserting 5:
25 10 5 20 30 28
Preorder display of AVL Tree after inserting 8:
25 10 5 8 20 30 28
Process returned 0 (0x0) execution time : 0.110 s
Press any key to continue.

```

3. Given the following binary tree:



- Mention height and depth of the each node.
- Represent given binary tree using linked list.
- Perform an In-order traversal and list the nodes visited.
- Perform a Pre-order traversal and list the nodes visited.
- Perform a Post-order traversal and list the nodes visited.
- Discuss a real-world application where each of these traversal methods would be most appropriate.

Soln:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
typedef struct Node
```

```
{
```

```

int data;

struct Node* left;

struct Node* right;

} Node;

Node* insert(Node* root, int data)
{
    Node* nw, *cur, *parent;
    nw = (Node*) malloc(sizeof(Node));
    if (nw)
    {
        nw->data = data;
        nw->left=nw->right=NULL;
        if (root == NULL)
            return nw;
        for(cur = root, parent = NULL; cur != NULL;)
        {
            parent = cur;
            if (nw->data < cur->data)
                cur = cur->left;
            else
                cur = cur->right;
        }
        if (nw->data < parent->data)
            parent->left = nw;
        else
            parent->right = nw;
    }
    else

```

```
{  
    printf("\nNode allocation failed\n");  
}  
return root;  
}
```

```
void preorder (struct Node *cur)
```

```
{  
    if (cur == NULL)  
        return;  
    printf ("%d ", cur -> data);  
    preorder (cur -> left);  
    preorder (cur -> right);  
}
```

```
void inorder (struct Node *cur)
```

```
{  
    if (cur == NULL)  
        return;  
    inorder (cur -> left);  
    printf ("%d ", cur -> data);  
    inorder (cur -> right);  
}
```

```
void postorder (struct Node *cur)
```

```
{  
    if (cur == NULL)  
        return;  
    postorder (cur -> left);
```

```
    postorder (cur -> right);
    printf ("%d ", cur -> data);
}

int main()
{
    int values[7] = {50, 30, 20, 40, 70, 60, 80}, i;

    Node* root = NULL;
    for (i = 0; i < 7; i++)
        root = insert(root, values[i]);

    printf("Inorder Display:\n");
    inorder(root);
    printf("\n\nPreorder Display:\n");
    preorder(root);
    printf("\n\nPostorder Display:\n");
    postorder(root);
    printf("\n\n");
    return 0;
}
```

```

57 }
58
59 void postorder (struct Node *cur)
60 {
61     if (cur == NULL)
62         return;
63     postorder (cur -> left);
64     postorder (cur -> right);
65     printf ("%d ", cur -> data);
66 }
67
68 int main()
69 {
70     int values[7] = {50, 30, 20, 40, 70, 80, 60};
71
72     Node* root = NULL;
73     for (i = 0; i < 7; i++)
74         root = insert(root, values[i]);
75
76     printf("Inorder Display:\n");
77     inorder(root);
78     printf("\n\nPreorder Display:\n");
79     preorder(root);
80     printf("\n\nPostorder Display:\n");
81     postorder(root);
82     printf("\n\n");
83     return 0;
84 }
85

```

Output:

```

Inorder Display:
20 30 40 50 60 70 80

Preorder Display:
50 30 20 40 70 60 80

Postorder Display:
20 40 30 60 80 70 50

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.

```

4. Consider the following unweighted, undirected graph:

Vertices: A, B, C, D, E, F Edges: (A, B), (A, C), (B, D), (C, E), (D, F), (E, F)

- Represent this graph using an Adjacency Matrix.
- Represent this graph using an Adjacency List.
- Starting from vertex A, perform a Breadth-First Search (BFS) and list the order in which nodes are visited.
- Starting from vertex A, perform a Depth-First Search (DFS) and list the order in which nodes are visited. (Assume alphabetical order for visiting adjacent unvisited nodes in both traversals).

Soln:

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

```

```

#define SIZE 20

```

```

int V, E, G[SIZE][SIZE], visited[SIZE];
char vertices[SIZE];

```

```

int getIndex(char c)
{
    for (int i = 0; i < V; i++)

```

```

    {
        if (vertices[i] == c)
            return i;
    }
    return -1;
}

void DFS(int i)
{
    visited[i] = 1;
    printf("%c -> ", vertices[i]);
    for (int j = 0; j < V; j++)
    {
        if (G[i][j] == 1 && visited[j] == 0)
            DFS(j);
    }
}

void BFS(int start)
{
    int queue[SIZE], front = 0, rear = 0, i, current;
    for (i = 0; i < V; i++)
        visited[i] = 0;

    queue[rear++] = start;
    visited[start] = 1;

    while (front < rear)
    {
        current = queue[front++];
        printf("%c -> ", vertices[current]);
        for (i = 0; i < V; i++)
        {
            if (G[current][i] == 1 && visited[i] == 0)
            {
                queue[rear++] = i;
                visited[i] = 1;
            }
        }
    }
}

int main()
{
    int i, j, idx1, idx2;
    char v1, v2, source;

```



```

printf("\t\tGraphs with Character Vertices\n");
printf("Enter number of vertices: ");
scanf("%d", &V);

printf("Enter labels for vertices (A-Z):\n");
for (i = 0; i < V; i++) {
    printf("Vertex %d: ", i + 1);
    scanf(" %c", &vertices[i]);
    vertices[i] = toupper(vertices[i]);
}
for (i = 0; i < V; i++)
    for (j = 0; j < V; j++)
        G[i][j] = 0;

printf("Enter number of edges: ");
scanf("%d", &E);

for (i = 0; i < E; i++)
{
    printf("Enter edge %d (format: A B): ", i + 1);
    scanf(" %c %c", &v1, &v2);
    idx1 = getIndex(toupper(v1));
    idx2 = getIndex(toupper(v2));
    if (idx1 != -1 && idx2 != -1)
    {
        G[idx1][idx2] = 1;
        G[idx2][idx1] = 1;
    }
    else
    {
        printf("Invalid vertex label.\n");
        i--;
    }
}

printf("\nAdjacency Matrix:\n");
printf(" ");
for (i = 0; i < V; i++)
    printf(" %c ", vertices[i]);
printf("\n");
for (i = 0; i < V; i++)
{
    printf(" %c |", vertices[i]);
    for (j = 0; j < V; j++)
        printf(" %d ", G[i][j]);
}

```

```

        printf("\n");
    }

    printf("\nEnter source vertex for DFS and BFS: ");
    scanf("%c", &source);
    int srcIndex = getIndex(toupper(source));
    if (srcIndex == -1)
    {
        printf("Invalid source vertex.\n");
        return 1;
    }

    printf("\nDFS Traversal: ");
    for (i = 0; i < V; i++) visited[i] = 0;
    DFS(srcIndex);

    printf("\nBFS Traversal: ");
    BFS(srcIndex);

    printf("\n");

    return 0;
}

```

The screenshot shows the Code::Blocks IDE with the following components:

- main.c**: The C source code for DFS and BFS, with line numbers 101 to 129 visible.
- Run Window**: Displays the program's execution output:


```

      Graphs with Character Vertices
      Enter number of vertices: 6
      Enter labels for vertices (A-Z):
      Vertex 1: A
      Vertex 2: B
      Vertex 3: C
      Vertex 4: D
      Vertex 5: E
      Vertex 6: F
      Enter number of edges: 6
      Enter edge 1 (format: A B): A B
      Enter edge 2 (format: A B): A C
      Enter edge 3 (format: A B): B D
      Enter edge 4 (format: A B): C E
      Enter edge 5 (format: A B): D F
      Enter edge 6 (format: A B): E F
      Adjacency Matrix:
      A B C D E F
      A | 0 1 1 0 0 0
      B | 1 0 0 1 0 0
      C | 1 0 0 0 1 0
      D | 0 1 0 0 0 1
      E | 0 0 1 0 0 1
      F | 0 0 0 1 1 0
      Enter source vertex for DFS and BFS: A
      DFS Traversal: A -> B -> D -> F -> E -> C ->
      BFS Traversal: A -> B -> C -> D -> E -> F ->
      Process returned 0 (0x0)   execution time : 53.271 s
      Press any key to continue.
      
```
- Taskbar**: Shows various open applications including Code::Blocks, Search results, Cccc, Build log, Build messages, CppCheck/Vera++, CppCheck/Vera++ messages, Cscope, Debugger, DoxyBlocks, and Fortn.

6. You have a hash table of size 10 (indices 0-9) and the hash function $h(k)=k \bmod 10$. Insert the following sequence of keys: [43, 22, 1, 31, 77, 99, 11, 55, 60]. Perform the insertions using the following collision resolution techniques:

- a) Chaining (using linked lists for collisions)
- b) Linear probing
- c) Quadratic probing

For each method, provide the final state of the hash table. In the case of chaining, include the linked lists representing collided elements at each index.

Soln:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#define SIZE 10
```

```
typedef struct
```

```
{
```

```
    int data;
```

```
    struct Node* next;
```

```
} Node;
```

```
Node* hashTableChain[SIZE] = {NULL};
```

```
int Linear[SIZE], Quadratic[SIZE];
```

```
int hash(int key)
```

```
{
```

```
    return key % SIZE;
```

```
}
```

```
void insertChaining(int key)
```

```
{
```

```

int index = hash(key);

Node* newNode = (Node*) malloc(sizeof(Node));

newNode->data = key;

newNode->next = hashTableChain[index];

hashTableChain[index] = newNode;
}

```

```

void displayChaining()
{
    Node *temp;

    int i;

    printf("\nChaining Hash Table:\n");
    for (i = 0; i < SIZE; i++)
    {
        printf("[%d]:", i);

        temp = hashTableChain[i];

        while (temp)
        {
            printf(" %d ->", temp->data);

            temp = temp->next;
        }

        printf(" NULL\n");
    }
}

```

```

void insertLinearProbing(int key)
{
    int index = hash(key), org = index;

    while (Linear[index] != -1)

```

```

{
    index = (index + 1) % SIZE;
    if (index == org)
    {
        printf("Hash table full! Could not insert %d in Linear Probing.\n", key);
        return;
    }
}
Linear[index] = key;
}

```

```

void displayLinearProbing()
{
    printf("\nLinear Probing Hash Table:\n");
    for (int i = 0; i < SIZE; i++)
        printf("[%d]: %d\n", i, Linear[i]);
}

```

```

void insertQuadraticProbing(int key)
{
    int index = hash(key), i = 0, newIndex;
    while (i < SIZE)
    {
        newIndex = (index + i*i) % SIZE;
        if (Quadratic[newIndex] == -1)
        {
            Quadratic[newIndex] = key;
            return;
        }
    }
}

```


```
        i++;
    }
    printf("Hash table full! Could not insert %d in Quadratic Probing.\n", key);
}
```

```
void displayQuadraticProbing()
{
    printf("\nQuadratic Probing Hash Table:\n");
    for (int i = 0; i < SIZE; i++)
        printf("[%d]: %d\n", i, Quadratic[i]);
}
```

```
int main() {
    int keys[9] = {43, 22, 1, 31, 77, 99, 11, 55, 60};

    for (int i = 0; i < SIZE; i++)
    {
        Linear[i] = -1;
        Quadratic[i] = -1;
    }
    for (int i = 0; i < 9; i++)
    {
        insertChaining(keys[i]);
        insertLinearProbing(keys[i]);
        insertQuadraticProbing(keys[i]);
    }
    displayChaining();
    displayLinearProbing();
    displayQuadraticProbing();
}
```

}

D:\KLE\Sem-4\DSA\hashing\main.c	C/C++	Windows (CR+LF)	WINDOWS-1252	Line 101, Col 6, Pos 2133	Insert	Read/Write	default	
---------------------------------	-------	-----------------	--------------	---------------------------	--------	------------	---------	---