26/2/24

Lab - 8

1) BST

ans)

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 20
struct queue
{
    int items [SIZE];
    int front;
    int rear;
};

struct queue* createqueue()
{
    struct queue* q = malloc (sizeof (struct queue));
    q -> front = -1;
    q -> rear = -1;
    return q;
}
void enqueue (struct queue* q, int value)
{
    if (q -> rear == SIZE-1)
        printf ("\n Queue is full");
    else
    {
        if (q -> front == -1)
            q -> front = 0;
        q -> rear ++;
        q -> items [q -> rear] = value;
}
int dequeue (struct queue* q)
{
```

26/2/24

```
int item;
if (q->rear == -1)
{
    printf("Queue is empty");
    item = -1;
}
else
{
    item = q->items[q->front];
    q->front++;
    if (q->front > q->rear)
    {
        printf("Resetting queue");
        q->front = q->rear = -1;
    }
}
return item;
}
void printqueue(struct queue* q)
{
    int i = q->front;
    if (q->rear == -1)
    {
        printf("Queue is empty");
    }
    else
    {
        printf("\n Queue contains \n");
        for (i = q->front; i < q->rear; i++)
        {
            printf("%d", q->items[i]);
        }
    }
}
```

```
struct node
{
    int vertex;
    struct node* next;
};
struct node* createnode(int);
struct Graph
{
    int numvertices;
    struct node** adjlists;
    int* visited;
};
void bfs(struct Graph* graph, int startvertex)
{
    struct queue* q = createqueue();
    graph->visited[startvertex] = 1;
    enqueue(q, startvertex);
    while(!(q->rear == -1))
    {
        printqueue(q);
        int currentvertex = dequeue(q);
        printf("Visited %d\n", currentvertex);
        struct node* temp = graph->adjlists[currentvertex];
        while(temp)
        {
            int adjvertex = temp->vertex;
            if(graph->visited[adjvertex] == 0)
            {
                graph->visited[adjvertex] = 1;
                enqueue(q, adjvertex);
            }
            temp = temp->next;
        }
    }
}
```

```
struct node* createnode (int v)
{
    struct node* newnode = malloc (sizeof (struct node));
    newnode -> Vertex = v;
    newnode -> next = NULL;
    return newnode;
}

struct Graph* creategraph (int vertices)
{
    struct Graph* graph = malloc (sizeof (struct Graph));
    graph -> numvertices = vertices;
    graph -> adjlists = malloc (vertices * sizeof (struct node));
    graph -> visited = malloc (vertices * sizeof (int));
    int i;
    for (i = 0; i < vertices; i++)
    {
        graph -> adjlists [i] = NULL;
        graph -> visited [i] = 0;
    }
    return graph;
}

void addedge (struct Graph* graph, int src, int dest)
{
    struct node* newnode = createnode (dest);
    newnode -> next = graph -> adjlists [src];
    graph -> adjlists [src] = newnode;
    newnode = createnode (src);
    newnode -> next = graph -> adjlists [dest];
    graph -> adjlists [dest] = newnode;
}

void main()
{
    int vertices, edges, src, dest;
    printf ("Enter the no. of vertices:");
```

```
scanf("%d", &vertices);
printf("Enter the no. of edges:");
scanf("%d", &edges);
struct Graph* graph = createGraph(vertices);
printf("Enter the edges (src, dest)\n");
for (int i=0; i<edges; i++)
{
    scanf("%d%d", &src, &dest);
    addEdge(graph, src, dest);
}
printGraph(graph);
bfs(graph, 0);
}
```

OUTPUT:

```
Enter the number of vertices: 5
Enter the number of edges: 4
Enter the edges (src, dest)
0 1
0 2
0 3
2 4
visited 0
visited 1
visited 2
visited 3
visited 4
```

26/2/24

```
Enter the number of vertices: 5
Enter the number of edges: 4
Enter edges (source destination):
0 1
0 2
1 3
2 4
Resetting queue Visited 0
Visited 2
Visited 1
Visited 4
Resetting queue Visited 3
```

2) DFS

ans)

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int vertex;
    struct node* next;
}
struct node* createnode(int v);
struct Graph
{
    int numvertices;
    int* visited;
    struct node** adjlists;
};
void dfs(struct Graph* graph, int vertex)
{
    struct node* adjlist = graph->adjlists[vertex];
    struct node* temp = adjlist;
    graph->visited[vertex] = 1;
    printf("visited %d \n", vertex);
    while(temp != NULL)
    {
        int connectedvertex = temp->vertex;
        if(graph->visited[connectedvertex] == 0)
        {
            dfs(graph, connectedvertex);
        }
        temp = temp->next;
    }
}
```

```
struct node* createnode (int v)
{
    struct node* newnode = malloc (sizeof (struct node));
    newnode -> vertex = v;
    newnode -> next = NULL;
    return newnode;
}

struct Graph* creategraph (int vertices)
{
    struct Graph* graph = malloc (sizeof (struct Graph));
    graph -> numvertices = vertices;
    graph -> adjlists = malloc (vertices * sizeof (struct node*));
    graph -> visited = malloc(vertices * sizeof (int));
    int i;
    for (i = 0; i < vertices ; i++)
    {
        graph -> adjlist[i] = NULL;
        graph -> visited [i] = 0;
    }
    return graph;
}

void addedge ( struct Graph* graph, int src, int dest)
{
    struct node* newnode = createnode (dest);
    newnode -> next = graph -> adjlists [src];
    graph -> adjlists [src] = newnode;
    newnode = createnode (src);
    newnode -> next = graph -> adjlists [dest];
    graph -> adjlists [dest] = newnode
}

void printgraph (struct Graph* graph)
{
```

```
    int v;
    for (v=0; v< graph->numvertices; v++)
    {
        struct node* temp = graph->adjlists[v];
        printf("\n Adjacency list of vertex %d\n", v);
        while (temp)
        {
            printf("%d ", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }
}

void main()
{
    int vertices, edges, src, dest;
    printf("Enter the no. of vertices:");
    scanf("%d", &vertices);
    printf("%d", &edges);
    struct Graph* graph = creategraph(vertices);
    printf("Enter the edges (src, dest)\n");
    for (int i=0; i< edges; i++)
    {
        scanf("%d %d", &src, &dest);
        addedge(graph, src, dest);
    }
    printgraph(graph);
    dfs(graph, 2);
}
```

OUTPUT:

```
Enter the no. of vertices: 4
Enter the no. of edges: 4
Enter the edges (src, dest)
```

0 1
0 2
1 2
2 3

Adjacency list of vertex 0
2 1

Adjacency list of vertex 1
2 0

Adjacency list of vertex 2
3 1 0

Adjacency list of vertex 3
2

visited 2
visited 3
visited 1
visited 0

```
Enter the number of vertices: 4
Enter the number of edges: 4
Enter edges (source destination):
0 1
0 2
1 2
2 3

 Adjacency list of vertex 0
 2 -> 1 ->

 Adjacency list of vertex 1
 2 -> 0 ->

 Adjacency list of vertex 2
 3 -> 1 -> 0 ->

 Adjacency list of vertex 3
 2 ->
Visited 2
Visited 3
Visited 1
Visited 0
```

1) Given a root node reference of a BST, delete the node with the given key in the BST. Return the root node reference (possibly updated) of the BST.

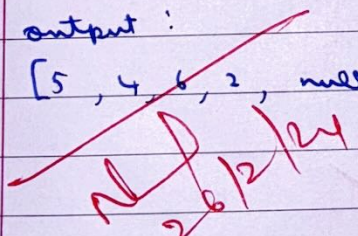
```
struct Treenode {
    int val;
    struct Treenode *left;
    struct Treenode *right;
};
struct Treenode * minvalnode (struct Treenode* node)
{
    struct Treenode* current = node;
    while (current && current -> left != NULL)
    {
        current = current -> left;
    }
    return current;
}
struct Treenode* deletenode struct Treenode* root,
int key)
{
    if (root == NULL)  return root;
    if (key < root -> val)
        root -> left = deletenode (root -> left, key);
    else if (key > root -> val)
        root -> right = deletenode (root -> right, key)
    else
    {
        if (root -> left == NULL)
        {
            struct Treenode* temp = root -> right;
            free (root);
            return temp;
        }
```

```
    else if (root -> right == NULL)
    {
        struct Treenode * temp = root -> left;
        free (root);
        return temp;
    }
    struct Treenode * temp = minvalnode (root -> right);
    root -> val = temp -> val;
    root -> right = deletenode ( root -> right, temp -> val);
  }
  return root;
}
```

OUTPUT:

Input:
root = [5, 3, 6, 2, 4, null, 7]
key = 3

output:
[5, 4, 6, 2, null, null, 7]



6) Given the root of a binary tree, return the leftmost value in the last row of the tree

ans)

```
struct Treenode
{
    int val;
    struct Treenode * left;
    struct Treenode * right;
};
int findbottomleftvalue ( struct Treenode * root)
{
    struct Treenode * queue [50];
```

</> Code

C ▾ Auto

```c
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
struct TreeNode* minValueNode(struct TreeNode* node) {
    struct TreeNode* current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}

struct TreeNode* deleteNode(struct TreeNode* root, int key) {
    if (root == NULL) return root;

    if (key < root->val)
        root->left = deleteNode(root->left, key);
    else if (key > root->val)
        root->right = deleteNode(root->right, key);
    else {
        if (root->left == NULL) {
            struct TreeNode* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;
        }

        struct TreeNode* temp = minValueNode(root->right);
        root->val = temp->val;
        root->right = deleteNode(root->right, temp->val);
    }
    return root;
}
```

Testcase | Test Result

Accepted Runtime: 2 ms

• Case 1 • Case 2 • Case 3

Input

root =

[5,3,6,2,4,null,7]

key =

3

Output

[5,4,6,2,null,null,7]

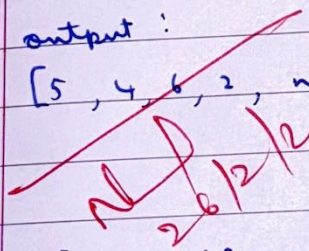
Expected

[5,4,6,2,null,null,7]

```
else if (root -> right == NULL)
{
    struct treenode * temp = root -> left;
    free (root);
    return temp;
}
struct treenode * temp = minvaluenode (root -> right);
root -> val = temp -> val;
root -> right = deletenode (root -> right, temp -> val);
}
return root;
}
```

OUTPUT:

input:
root = [5, 3, 6, 2, 4, null, 7]
key = 3
output:
[5, 4, 6, 2, null, null, 7]



4) Given the root of a binary tree, return the leftmost value in the last row of the tree

ans)

```
struct Treenode
{
    int val;
    struct Treenode * left;
    struct Treenode * right;
};
int findbottomleftvalue (struct Treenode * root)
{
    struct Treenode * queue [50];
```

```
int front = 0;
int rear = 0;
queue[rear++] = root;
int leftmostvalue = root->val;

while (front < rear)
{
    int levelsize = rear - front;
    for (int i=0; i< levelsize; i++)
    {
        struct treenode* current = queue[front++];
        if (i == 0)
            leftmostvalue = current->val;
        if (current->left != NULL)
            queue[rear++] = current->left;
        if (current->right != NULL)
            queue[rear++] = current->right;
    }
}
return leftmostvalue;
}
```

OUTPUT:

Input:

root = [2, 1, 3]

output:

1

26/2/24

</> Code

C ▾ Auto

```c
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
int findBottomLeftValue(struct TreeNode* root) {
    struct TreeNode* queue[10000]; // Assuming maximum 10000 nodes
    int front = 0, rear = 0;
    queue[rear++] = root;
    int leftmostValue = root->val;

    while (front < rear) {
        int levelSize = rear - front;
        for (int i = 0; i < levelSize; i++) {
            struct TreeNode* current = queue[front++];
            if (i == 0) // first node at the current level
                leftmostValue = current->val;
            if (current->left != NULL)
                queue[rear++] = current->left;
            if (current->right != NULL)
                queue[rear++] = current->right;
        }
    }

    return leftmostValue;
}
```

Testcase | Test Result

Accepted Runtime: 5 ms

• Case 1 • **Case 2**

Input

root =

```
[1,2,3,4,null,5,6,null,null,7]
```

Output

```
7
```

Expected

```
7
```