Date: 26/02/24

Week-7

1) BFS

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 40

struct queue{
        int items[SIZE];
        int front;
        int rear;
   };

  struct queue *createQueue();
   void enqueue(struct queue *q, int);
   int dequeue(struct queue *q);
   void display(struct queue *q);
   int isEmpty(struct queue *q);
   void printQueue(struct queue *q);

   struct node{
        int vertex;
        struct node *next;
    };

   struct node *createNode(int);

   struct Graph {
         int numVertices;
         struct node** adjLists;
         int *visited;
     };
```

```
// BFS algorithm
void bfs(struct Graph *graph, int startVertex){
    struct queue *q = createQueue();

    graph->visited[startVertex] = 1;
    enqueue(q, startVertex);

    while(!isEmpty(q)){
        printQueue(q);
        int currentVertex = dequeue(q);
        printf("Visited %d\n", currentVertex);

        struct node *temp = graph->adjLists[curentVertex];
        while(temp){
            int adjVertex = temp->vertex;

            if(graph->visited[adjVertex] == 0){
                graph->visited[adjVertex] = 1;
                enqueue(q, adjVertex);
            }
            temp = temp->next;
        }
    }
}

// creating a graph
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 * size of (int));

      int i;
    for (i=0 , i< vertices; i++) {
       graph -> adjLists [i] = NULL;
       graph -> visited [i] = 0;
    }
      return graph
  }

//add edge
 void addEdge (struct Graph *graph , int src , int dest){
   //add edge from src to dest
     struct node * newNode = createNode (dest);
      newNode -> Next = graph -> adjLists [src];
      graph -> adjLists [src] = newNode ;

   //add edge from dest to src
    newNode = createNode (src);
     newNode -> next = graph -> adjLists [dest];
     graph -> adjLists [dest] = newNode;
  }
  //create a queue
   struct queue * createQueue (){
        struct queue *q = malloc (sizeof (struct queue));
         q -> front = -1
         q -> rear = -1
         return q;
    }
```

```
//check if the queue is empty
int isEmpty(struct queue * q){
    if(q->rear == -1){
        return 1;
    else
        return 0;
  }

//adding elements into queue
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;
    }
}

//Removing elements from queue
int dequeue(struct queue *q){
    int item;
    if(isEmpty(q)){
        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;
}

//Print the Queue
void printQueue(struct queue *q){
        int i = q->front;
        if(isEmpty(q)){
            printf("Queue is empty");
        }else{
            printf("\nQueue contains \n");
            for(i=q->front; i<q->rear+1; i++){
                printf("%d", q->items[i]);
            }
        }
    }

int main(){
    struct Graph* grap = createGraph(6);
        addEdge(graph,0,1);
        addEdge(graph,0,2);
        addEdge(graph,1,2);
        addEdge(graph,1,4);
        addEdge(graph,1,3);
        addEdge(graph,2,4);
        addEdge(graph,3,4);

        bfs(graph,0);

        return 0;
    }
```

o/p:

```
Queue contains
 0 Resetting queue visited 0

 Queue contains
  2 1 visited 2

 Queue contains
  1 4 visited 1

 Queue contains
  4 3 visited 4

 Queue contains
  3 Resetting queue Visited 3
```

2) DFS

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

 struct node {
   int vertex;
   struct node * next;
 };

struct node * createNode (int v);

 struct Graph {
     int numVertices;
     int * visited;

   # we need int** to store a two dimensional array
  // similary. we need struct node ** to store an array of
    Linked Lists
     struct node** adjLists;
 }
```

```
//DFS algo

void DFS(struct Graph* graph, int vertor){
        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;
        }
   }


//create a Node

  struct node * createNode(int v){
        struct node* newNode = 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;
    }
```

```
//add edge
void addEdge(struct Graph* graph , int src, int dest){
  //add edge from src to dest
   struct node * NewNode = createNode(dest);
   newNode->next = graph->adjLists[src];
   graph->adjLists [src] = newNode;

   //add edge from dest to src
   newNode = createNode (src);
     newNode->next = graph->adjLists[dest];
      graph->adjLists [dest] = newNode ;
   }

  //print the graph
 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;
      }
      print("\n");
    }
  }
```

```
int main(){
    struct Graph * graph = createGraph(4);
        addEdge(graph, 0,1);
        addEdge(graph,0,2);
        addEdge(graph, 1,2);
        addEdge(graph,2,3);

        printGraph(graph);

        DFS(graph,2);

        return 0;
    }
```

O/p:

```
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
```

Leetcode - Delete a node in BST

```
struct TreeNode* Smallest(struct TreeNode* root)
{ struct TreeNode *cur = root;
  while(cur->left != NULL)
  { cur = cur->left;
  }
  return cur;
}

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 = smallest(root->right);
 root->val = temp->val;
  root->right = deleteNode(root->right, root->val);
 }
return root;
}
```

4) Leetcode - Bottom left tree value

```
typedef struct TreeNode TreeNode;
 #define MAX-NODE (10000);
 int findBottomLeftValue(const TreeNode* const pRoot){
     assert(pRoot != NULL);
      int firstValInRow;
      const TreeNode *bfsQueue[MAX_NODE];
      int get = 0, set = 0;

      bfsQueue[set] = pRoot;
       set += 1;

      do {
        firstValInRow = bfsQueue[get]->val;
for (int rest = set - get; rest > 0; rest -= 1) {
    const TreeNode * const pCur = bfsQueue[get];
    get += 1;

    if ( pCur->left != NULL) {
        bfsQueue [set] = pCur->left;
        set += 1;
    }
    if (pCur->right != NULL){
        bfsQueue [set] = pCur->right;
        set += 1;
    }
  }
}while (get < set);

return firstValInRow;
}
```

3) O/p:

Case1:

I/p:

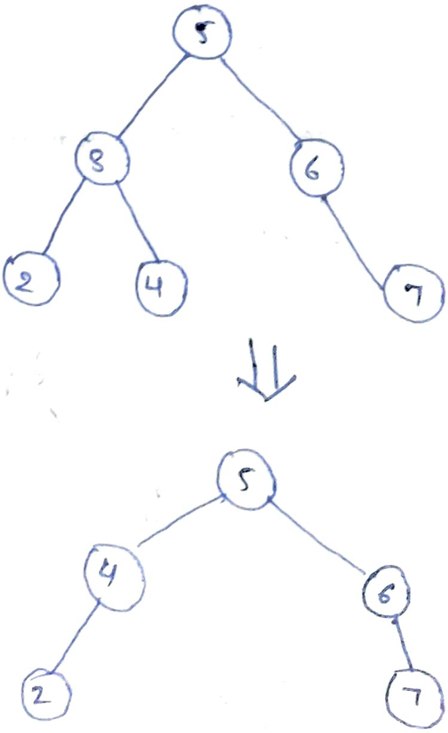
[5,3,6,2,4,null,7]

key =

3

O/p:

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



Case 2 :

Input :

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

key = 0

O/p :

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

Case 3 :

i/p : [ ]

key = 0

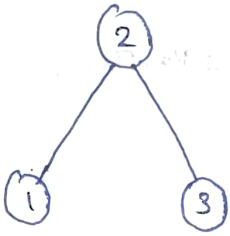
o/p : [ ]

4) O/p :

Case 1 :

root = [2,1,3]

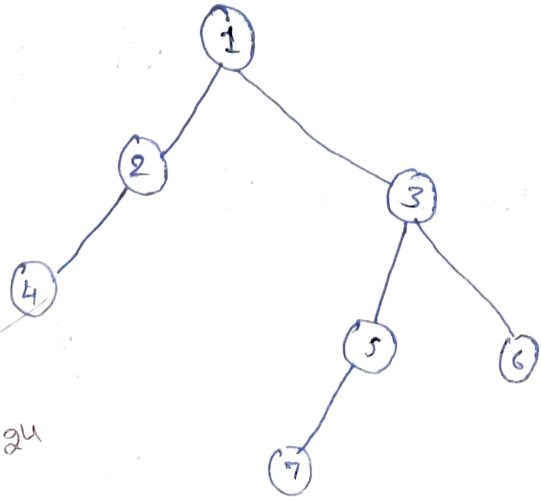
output = 1



case 2 :

I/p : [1,2,3,4,null,5,6,null,null,7]

o/p : 7



26.02.24