/*Q2. Write a C program to find union (of two linked lists) based on their information field that implements singly linked list (with information field Emp_Id and Name of employee for each node).

Name: Raj Basnet

Section: A2   Rollno. : 49

Course : B.Tech*/

Source Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>


typedef struct node {

   int emp_id;

   char *name;

   struct node *next;

} node;


// Function to create a new node

node* create_node(int emp_id, char *name) {

   node *temp = (node*)malloc(sizeof(node));

   temp->emp_id = emp_id;

   temp->name = (char*)malloc((strlen(name) + 1) * sizeof(char));

   strcpy(temp->name, name);

   temp->next = NULL;

   return temp;

}
```

```c
// Function to insert a new node at the end of the list
void insert(node **head, int emp_id, char *name) {
    node *temp = create_node(emp_id, name);
    if (*head == NULL) {
        *head = temp;
    } else {
        node *last = *head;
        while (last->next != NULL) {
            last = last->next;
        }
        last->next = temp;
    }
}

// Function to check if a node is present in the list
bool ispresent(node *head, int emp_id, char *name) {
    while (head != NULL) {
        if (head->emp_id == emp_id && strcmp(head->name, name) == 0) {
            return true;
        }
        head = head->next;
    }
    return false;
}
```

```
// Function to get the union of two linked lists
node* unionlists(node **head1, node **head2) {
   node *unionList = NULL;
   node *current = *head1;

   // Add all nodes from the first list
   while (current != NULL) {
      if (!ispresent(unionList, current->emp_id, current->name)) {
         insert(&unionList, current->emp_id, current->name);
      }
      current = current->next;
   }

   // Add all nodes from the second list
   current = *head2;
   while (current != NULL) {
      if (!ispresent(unionList, current->emp_id, current->name)) {
         insert(&unionList, current->emp_id, current->name);
      }
      current = current->next;
   }

   return unionList;
}

// Function to print the linked list
```

```c
void printList(node *head) {
    while (head != NULL) {
        printf("Emp_ID: %d, Name: %s -> ", head->emp_id, head->name);
        head = head->next;
    }
    printf("NULL\n");
}

// Main function for menu-driven operations
int main() {
    node *list1 = NULL;
    node *list2 = NULL;
    node *unionList = NULL;

    int choice, emp_id;
    char name[100];
     printf("\nMenu:\n");
        printf("1. Insert into List 1\n");
        printf("2. Insert into List 2\n");
        printf("3. Display List 1\n");
        printf("4. Display List 2\n");
        printf("5. Find Union\n");
        printf("6. Display Union\n");
        printf("7. Exit\n");
    while (1) {
```

```
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
  case 1:
    printf("Enter Employee ID: ");
    scanf("%d", &emp_id);
    printf("Enter Employee Name: ");
    scanf("%s", name);
    insert(&list1, emp_id, name);
    break;
  case 2:
    printf("Enter Employee ID: ");
    scanf("%d", &emp_id);
    printf("Enter Employee Name: ");
    scanf("%s", name);
    insert(&list2, emp_id, name);
    break;
  case 3:
    printf("List 1: ");
    printList(list1);
    break;
  case 4:
    printf("List 2: ");
    printList(list2);
    break;
```

```c
        case 5:
          unionList = unionlists(&list1, &list2);
          printf("Union of List 1 and List 2 has been created.\n");
          break;
        case 6:
          printf("Union List: ");
          printList(unionList);
          break;
        case 7:
          exit(0);
        default:
          printf("Invalid choice! Please enter a valid option.\n");
      }
    }
    return 0;
}
```

Output:

PS D:\c++\raj basnet> gcc q2.c

PS D:\c++\raj basnet> ./a.exe

Menu:

1. Insert into List 1

2. Insert into List 2

3. Display List 1

4. Display List 2

5. Find Union

6. Display Union

7. Exit

Enter your choice: 1

Enter Employee ID: 567

Enter Employee Name: raj

Enter your choice: 1

Enter Employee ID: 345

Enter Employee Name: sujal

Enter your choice: 1

Enter Employee ID: 678

Enter Employee Name: akshat

Enter your choice: 2

Enter Employee ID: 897

Enter Employee Name: raj

Enter your choice: 2

Enter Employee ID: 456

Enter Employee Name: akshat

Enter your choice: 2

Enter Employee ID: 345

Enter Employee Name: sujal

Enter your choice: 3

List 1: Emp_ID: 567, Name: raj -> Emp_ID: 345, Name: sujal -> Emp_ID: 678, Name: akshat -> NULL

Enter your choice: 4

List 2: Emp_ID: 897, Name: raj -> Emp_ID: 456, Name: akshat -> Emp_ID: 345, Name: sujal -> NULL

Enter your choice: 5

Union of List 1 and List 2 has been created.

Enter your choice: 6

Union List: Emp_ID: 567, Name: raj -> Emp_ID: 345, Name: sujal -> Emp_ID: 678, Name: akshat -> Emp_ID: 897, Name: raj -> Emp_ID: 456, Name: akshat -> NULL

Enter your choice: 7

/*Q3. Write a C program to create a linked list P, then write a ‘C’ function named split to create two linked lists Q & R from P So that Q contains all elements in odd positions of P and R contains the remaining elements. Finally print both linked lists i.e. Q and R.

Name: Raj Basnet

Section: A2   Rollno. : 49

Course : B.Tech*/

Source Code:

```
#include <stdio.h>

#include <stdlib.h>


// Definition for the Node structure

typedef struct Node {

   int data;

   struct Node* next;

} Node;


// Function to create a new node

Node* create_node(int data) {

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

   temp->data = data;

   temp->next = NULL;

   return temp;

}


// Function to insert a node at the end of the list

void insert(Node** head, int data) {

   Node* temp = create_node(data);
```

```
    if (*head == NULL) {
        *head = temp;
        return;
    }
    Node* temp2 = *head;
    while (temp2->next) {
        temp2 = temp2->next;
    }
    temp2->next = temp;
}

// Function to split the list into two based on even and odd positions
void split(Node** P, Node** Q, Node** R) {
    if ((*P) == NULL) {
        return;
    }
    int index = 1;
    Node* current = *P;
    while (current) {
        if (index % 2 == 0) {
            insert(Q, current->data);
        } else {
            insert(R, current->data);
        }
        current = current->next;
        index++;
```

```
    }
}

// Function to display the list
void display(Node* head) {
    while (head) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

// Main function for menu-driven operations
int main() {
    Node* P = NULL; // Original list
    Node* Q = NULL; // List for even positions
    Node* R = NULL; // List for odd positions
    int choice, data;
    printf("\nMenu:\n");
        printf("1. Insert into original list\n");
        printf("2. Display original list\n");
        printf("3. Split the list\n");
        printf("4. Display even-position list\n");
        printf("5. Display odd-position list\n");
        printf("6. Exit\n");
    while (1) {
```

```
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
  case 1:
    printf("Enter data to insert: ");
    scanf("%d", &data);
    insert(&P, data);
    break;
  case 2:
    printf("Original list: ");
    display(P);
    break;
  case 3:
    split(&P, &Q, &R);
    printf("List has been split into even and odd positions.\n");
    break;
  case 4:
    printf("Even-position list: ");
    display(Q);
    break;
  case 5:
    printf("Odd-position list: ");
    display(R);
    break;
```

```c
        case 6:
          exit(0);
        default:
          printf("Invalid choice! Please enter a valid option.\n");
      }
    }
    return 0;
}
```

Output:

PS D:\c++\raj basnet> gcc q3.c

PS D:\c++\raj basnet> ./a.exe

Menu:

1. Insert into original list

2. Display original list

3. Split the list

4. Display even-position list

5. Display odd-position list

6. Exit

Enter your choice: 1

Enter data to insert: 67

Enter your choice: 1

Enter data to insert: 78

Enter your choice: 1

Enter data to insert: 89

Enter your choice: 1

Enter data to insert: 45

Enter your choice: 1

Enter data to insert: 34

Enter your choice: 1

Enter data to insert: 23

Enter your choice: 1

Enter data to insert: 12

Enter your choice: 2

Original list: 67 -> 78 -> 89 -> 45 -> 34 -> 23 -> 12 -> NULL

Enter your choice: 3

List has been split into even and odd positions.

Enter your choice: 4

Even-position list: 78 -> 45 -> 23 -> NULL

Enter your choice: 5

Odd-position list: 67 -> 89 -> 34 -> 12 -> NULL

Enter your choice: 6

/* Q4. Write a C program to create a binary search tree and perform following operations:

1) Find node having smallest data in the BST.

2) Delete a node from the tree.

3) Find total number nodes having common parent.

4) Find height of a binary search tree

5) Count total numbers of nodes from right hand side of root node

Name: Raj Basnet

Section: A2 Rollno,: 49

Course : B.Tech*/

Source Code:

```
#include <stdio.h>

#include <stdlib.h>


// Definition for a binary tree node.

typedef struct tree {

   int data;

   struct tree* left;

   struct tree* right;

} tree;


// Function to create a new node with given data.

tree* create_tree(int data) {

   tree* temp = (tree*)malloc(sizeof(tree));

   temp->data = data;

   temp->left = NULL;

   temp->right = NULL;
```

```
    return temp;
}

// Function to insert a new node with given key in the BST.
tree* insert(tree* root, int data) {
    if (root == NULL) {
        return create_tree(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}

// Function to find the inorder predecessor of a node.
int inorderprec(tree* root) {
    tree* curr = root->right;
    while (curr && curr->left != NULL) {
        curr = curr->left;
    }
    return curr->data;
}

// Function to delete a node with given key in the BST.
```

```
tree* delete_(tree* root, int data) {
  if (root == NULL) {
    return NULL;
  }
  if (data < root->data) {
    root->left = delete_(root->left, data);
  } else if (data > root->data) {
    root->right = delete_(root->right, data);
  } else {
    if (root->left == NULL && root->right == NULL) {
      free(root);
      return NULL;
    } else if (root->left == NULL) {
      tree* temp = root->right;
      free(root);
      return temp;
    } else if (root->right == NULL) {
      tree* temp = root->left;
      free(root);
      return temp;
    } else {
      int x = inorderprec(root);
      root->data = x;
      root->right = delete_(root->right, x);
    }
  }
```

```
    return root;
}

// Function to find the height of the BST.
int height(tree* root) {
    if (root == NULL) {
        return 0;
    }
    int leftheight = height(root->left);
    int rightheight = height(root->right);
    return (leftheight > rightheight ? leftheight : rightheight) + 1;
}

// Function to find the smallest element in the BST.
int smallest(tree* root) {
    while (root && root->left) {
        root = root->left;
    }
    return root ? root->data : -1;  // Return -1 if the tree is empty.
}

// Function to find nodes with both left and right children.
void find_common_parent(tree* root, int* c) {
    if (root == NULL) {
        return;
    }
```

```c
    if (root->left != NULL && root->right != NULL) {
        printf("Parent Node: %d, ", root->data);
        printf("Left Child: %d, ", root->left->data);
        printf("Right Child: %d\n", root->right->data);
        (*c)++;
    }
    find_common_parent(root->left, c);
    find_common_parent(root->right, c);
}

// Function to count nodes on the right-hand side.
void right_hand(tree* root, int* d) {
    while (root->right != NULL) {
        root = root->right;
        (*d)++;
    }
}

// Inorder traversal of the BST.
void inorder(tree* root) {
    if (root == NULL) {
        return;
    }
    inorder(root->left);
    printf("%d  ", root->data);
    inorder(root->right);
```

```
}

int main() {
    tree* root = NULL;
    int choice, data, count = 0, depth = 0;

    printf("\n\nMenu:\n");
    printf("1. Insert\n");
    printf("2. Delete\n");
    printf("3. Find Height\n");
    printf("4. Find Common Parent Nodes\n");
    printf("5. Count Nodes on Right-hand Side\n");
    printf("6. Find Smallest Element\n");
    printf("7. Exit\n");

    while (1) {
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                root = insert(root, data);
                printf("Node inserted.\n");
                printf("Inorder traversal: ");
                inorder(root);
```

```
  printf("\n");
  break;

case 2:
  printf("Enter data to delete: ");
  scanf("%d", &data);
  root = delete_(root, data);
  printf("Node deleted.\n");
  printf("Inorder traversal: ");
  inorder(root);
  printf("\n");
  break;

case 3:
  printf("Height of the BST: %d\n", height(root));
  break;

case 4:
  count = 0;
  find_common_parent(root, &count);
  printf("Total number of nodes with both left and right children: %d\n", count);
  break;

case 5:
  depth = 0;
  right_hand(root, &depth);
```

```
            printf("Total number of nodes on the right-hand side: %d\n", depth);
            break;

        case 6:
            printf("Smallest element in the BST: %d\n", smallest(root));
            break;

        case 7:
            exit(0);

        default:
            printf("Invalid choice! Please enter a valid option.\n");
        }
    }
    return 0;
}
```

OUTPUT:

PS D:\c++\raj basnet> gcc q4.c

PS D:\c++\raj basnet> ./a.exe

Menu:

1. Insert

2. Delete

3. Find Height

4. Find Common Parent Nodes

5. Count Nodes on Right-hand Side

6. Find Smallest Element

7. Exit

Enter your choice: 1

Enter data to insert: 56

Node inserted.

Inorder traversal: 56

Enter your choice: 1

Enter data to insert: 67

Node inserted.

Inorder traversal: 56 67

Enter your choice: 1

Enter data to insert: 78

Node inserted.

Inorder traversal: 56 67 78

Enter your choice: 1

Enter data to insert: 89

Node inserted.

Inorder traversal: 56 67 78 89

Enter your choice: 1

Enter data to insert: 89

Node inserted.

Inorder traversal: 56 67 78 89

Enter your choice: 1

Enter data to insert: 89

Node inserted.

Inorder traversal: 56 67 78 89

Enter your choice: 1

Enter data to insert: 56

Node inserted.

Inorder traversal: 56 67 78 89

Enter your choice: 1

Enter data to insert: 34

Node inserted.

Inorder traversal: 34 56 67 78 89

Enter your choice: 1

Enter data to insert: 23

Node inserted.

Inorder traversal: 23 34 56 67 78 89

Enter your choice: 1

Enter data to insert: 12

Node inserted.

Inorder traversal: 12 23 34 56 67 78 89

Enter your choice: 3

Height of the BST: 4

Enter your choice: 2

Enter data to delete: 34

Node deleted.

Inorder traversal: 12 23 56 67 78 89

Enter your choice: 6

Smallest element in the BST: 12

Enter your choice: 5

Total number of nodes on the right-hand side: 3

Enter your choice: 4

Parent Node: 56, Left Child: 23, Right Child: 67

Total number of nodes with both left and right children: 1

Enter your choice: 7

/* Q5. Write a C program to implement Kurskal's algorithm to find minimal spanning tree from a given graph.

Name: Raj Basnet

Section: A2 Rollno,: 49

Course : B.Tech*/

Source Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


// Definition for the Union-Find structure

typedef struct UnionFind {

   int *parent;

   int *rank;

} UnionFind;


// Function to create a Union-Find structure

UnionFind* create_union(int n) {

   UnionFind* u = (UnionFind*)malloc(sizeof(UnionFind));

   u->parent = (int*)malloc(n * sizeof(int));

   u->rank = (int*)malloc(n * sizeof(int));

   for (int i = 0; i < n; i++) {

      u->parent[i] = i;

      u->rank[i] = 0;

   }

   return u;
```

```c
}

// Function to find the representative of the set containing i
int find(UnionFind *u, int i) {
  if (u->parent[i] != i) {
    u->parent[i] = find(u, u->parent[i]);
  }
  return u->parent[i];
}

// Function to union two sets containing u and v
void union_sets(UnionFind *u, int u_set, int v_set) {
  int root_u = find(u, u_set);
  int root_v = find(u, v_set);
  if (root_u != root_v) {
    if (u->rank[root_u] > u->rank[root_v]) {
      u->parent[root_v] = root_u;
    } else if (u->rank[root_u] < u->rank[root_v]) {
      u->parent[root_u] = root_v;
    } else {
      u->parent[root_u] = root_v;
      u->rank[root_v]++;
    }
  }
}
```

```
// Definition for graph node
typedef struct graph {
   int data;
   int wt;
   struct graph* next;
} graph;

// Definition for edge node
typedef struct edge {
   int src;
   int des;
   int wt;
   struct edge* next;
} edge;

// Function to create a new graph node
graph* create_graph_node(int data, int wt) {
   graph* temp = (graph*)malloc(sizeof(graph));
   temp->data = data;
   temp->wt = wt;
   temp->next = NULL;
   return temp;
}

// Function to create a new edge node
edge* create_edge_node(int src, int des, int wt) {
```

```
    edge* temp = (edge*)malloc(sizeof(edge));
    temp->src = src;
    temp->des = des;
    temp->wt = wt;
    temp->next = NULL;
    return temp;
}

// Function to add a node into the graph in sorted order based on weight
void addEdgeSort(edge** head, int src, int wt, int des) {
    edge* temp = create_edge_node(src, des, wt);

    // Check if the list is empty or if the new node should be the first node
    if (*head == NULL || (*head)->wt > temp->wt) {
        temp->next = *head;
        *head = temp;
        return;
    }

    // Find the correct position to insert the new node
    edge* temp2 = *head;
    while (temp2->next != NULL && temp2->next->wt < temp->wt) {
        temp2 = temp2->next;
    }

    // Insert the new node at the correct position
```

```
    temp->next = temp2->next;
    temp2->next = temp;
}

// Function to insert a node into the graph
void insert(graph** head, int data, int wt) {
    graph* temp = create_graph_node(data, wt);
    if (*head == NULL) {
        *head = temp;
        return;
    }
    graph* temp2 = *head;
    while (temp2->next) {
        temp2 = temp2->next;
    }
    temp2->next = temp;
}

// Function to read the graph
void readGraph(graph* Graph[], int n) {
    for (int i = 0; i < n; i++) {
        int k;
        printf("Enter the number of edges adjacent to vertex %d: ", i);
        scanf("%d", &k);
        for (int j = 0; j < k; j++) {
            int y;
```

```
            int wt;
            printf("Enter the adjacent vertex: ");
            scanf("%d", &y);
            printf("Enter its weight: ");
            scanf("%d", &wt);
            insert(&Graph[i], y, wt);
        }
    }
}

// Function to display the graph
void displayGraph(graph* Graph[], int n) {
    for (int i = 0; i < n; i++) {
        printf("Vertex %d: ", i);
        graph* temp = Graph[i];
        while (temp) {
            printf("%d (%d) -> ", temp->data, temp->wt);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

// Function to add edges from the adjacency list to a sorted edge list
void addGraphEdges(graph* Graph[], int n, edge** head) {
    for (int i = 0; i < n; i++) {
```

```
        graph* p = Graph[i];
        while (p) {
            addEdgeSort(head, i, p->wt, p->data);
            p = p->next;
        }
    }
}

// Function to display the edges in the sorted list
void displayEdges(edge* head) {
    while (head) {
        printf("Source: %d, Destination: %d, Weight: %d -> ", head->src, head->des, head->wt);
        head = head->next;
    }
    printf("NULL\n");
}

// Function to find the Minimum Spanning Tree (MST) using Kruskal's algorithm
void kruskal(edge* edgeList, int n) {
    UnionFind* u = create_union(n);
    edge* mst = NULL;
    int edges_added = 0; // Counter for the number of edges added to the MST

    while (edgeList && edges_added < n - 1) {
        int u_set = find(u, edgeList->src);
        int v_set = find(u, edgeList->des);
```

```
        if (u_set != v_set) {
            addEdgeSort(&mst, edgeList->src, edgeList->wt, edgeList->des);
            union_sets(u, u_set, v_set);
            edges_added++; // Increment the number of edges added to the MST
        }

        edgeList = edgeList->next;
    }

    printf("Minimum Spanning Tree (MST):\n");
    displayEdges(mst);
}

// Main function for menu-driven operations
int main() {
    int n, choice;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    graph* Graph[n];
    for (int i = 0; i < n; i++) {
        Graph[i] = NULL;
    }

    edge* edgeList = NULL;
```

```
printf("\nMenu:\n");
    printf("1. Read Graph\n");
    printf("2. Display Graph\n");
    printf("3. Sort Edges\n");
    printf("4. Display Edges\n");
    printf("5. Find MST using Kruskal's Algorithm\n");
    printf("6. Exit\n");
while (1) {

    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            readGraph(Graph, n);
            break;
        case 2:
            displayGraph(Graph, n);
            break;
        case 3:
            edgeList = NULL; // Reset edge list before sorting
            addGraphEdges(Graph, n, &edgeList);
            break;
        case 4:
            displayEdges(edgeList);
            break;
```

```c
        case 5:
          kruskal(edgeList, n);
          break;
        case 6:
          exit(0);
        default:
          printf("Invalid choice! Please enter a valid option.\n");
      }
    }
    return 0;
}
```

Output:

PS D:\c++\raj basnet> gcc q5.c

PS D:\c++\raj basnet> ./a.exe

Enter the number of vertices: 3

Menu:

1. Read Graph

2. Display Graph

3. Sort Edges

4. Display Edges

5. Find MST using Kruskal's Algorithm

6. Exit

Enter your choice: 1

Enter the number of edges adjacent to vertex 0: 1

Enter the adjacent vertex: 2

Enter its weight: 2

Enter the number of edges adjacent to vertex 1: 2

Enter the adjacent vertex: 1

Enter its weight: 2

Enter the adjacent vertex: 2

Enter its weight: 3

Enter the number of edges adjacent to vertex 2: 4

Enter the adjacent vertex: 1

Enter its weight: 2

Enter the adjacent vertex: 1

Enter its weight: 3

Enter the adjacent vertex: 2

Enter its weight: 4

Enter the adjacent vertex: 3

Enter its weight: 6

Enter your choice: 2

Vertex 0: 2 (2) -> NULL

Vertex 1: 1 (2) -> 2 (3) -> NULL

Vertex 2: 1 (2) -> 1 (3) -> 2 (4) -> 3 (6) -> NULL

Enter your choice: 3

Enter your choice: 4

Source: 0, Destination: 2, Weight: 2 -> Source: 2, Destination: 1, Weight: 2 -> Source: 1, Destination: 1, Weight: 2 -> Source: 2, Destination: 1, Weight: 3 -> Source: 1, Destination: 2, Weight: 3 -> Source: 2, Destination: 2, Weight: 4 -> Source: 2, Destination: 3, Weight: 6 -> NULL

Enter your choice: 5

Minimum Spanning Tree (MST):

Source: 0, Destination: 2, Weight: 2 -> Source: 2, Destination: 1, Weight: 2 -> NULL

Enter your choice: 6

/* Q8. Write a C program to store the details of a weighted graph (Using linked list).

Name: Raj Basnet

Section : A2 Rollno.: 49

Course : B.Tech*/

Source Code:

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

// Definition for graph node
typedef struct graph {
   int data;
   int wt;
   struct graph* next;
} graph;

// Definition for edge node
typedef struct edge {
   int src;
   int des;
   int wt;
   struct edge* next;
} edge;

// Function to create a new graph node
graph* create_graph_node(int data, int wt) {
```

```
    graph* temp = (graph*)malloc(sizeof(graph));
    temp->data = data;
    temp->wt = wt;
    temp->next = NULL;
    return temp;
}

// Function to create a new edge node
edge* create_edge_node(int src, int des, int wt) {
    edge* temp = (edge*)malloc(sizeof(edge));
    temp->src = src;
    temp->des = des;
    temp->wt = wt;
    temp->next = NULL;
    return temp;
}

// Function to add a node into the graph in sorted order based on weight
void addEdgeSort(edge** head, int src, int wt, int des) {
    edge* temp = create_edge_node(src, des, wt);

    // Check if the list is empty or if the new node should be the first node
    if (*head == NULL || (*head)->wt > temp->wt) {
        temp->next = *head;
        *head = temp;
        return;
```

```
    }

    // Find the correct position to insert the new node
    edge* temp2 = *head;
    while (temp2->next != NULL && temp2->next->wt < temp->wt) {
        temp2 = temp2->next;
    }

    // Insert the new node at the correct position
    temp->next = temp2->next;
    temp2->next = temp;
}

// Function to insert a node into the graph
void insert(graph** head, int data, int wt) {
    graph* temp = create_graph_node(data, wt);
    if (*head == NULL) {
        *head = temp;
        return;
    }
    graph* temp2 = *head;
    while (temp2->next) {
        temp2 = temp2->next;
    }
    temp2->next = temp;
}
```

```
// Function to read the graph
void readGraph(graph* Graph[], int n) {
  for (int i = 0; i < n; i++) {
    int k;
    printf("Enter the number of edges adjacent to vertex %d: ", i);
    scanf("%d", &k);
    for (int j = 0; j < k; j++) {
      int y;
      int wt;
      printf("Enter the adjacent vertex: ");
      scanf("%d", &y);
      printf("Enter its weight: ");
      scanf("%d", &wt);
      insert(&Graph[i], y, wt);
    }
  }
}

// Function to display the graph
void displayGraph(graph* Graph[], int n) {
  for (int i = 0; i < n; i++) {
    printf("Vertex %d: ", i);
    graph* temp = Graph[i];
    while (temp) {
      printf("%d (%d) -> ", temp->data, temp->wt);
```

```
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

// Function to add edges from the adjacency list to a sorted edge list
void addGraphEdges(graph* Graph[], int n, edge** head) {
    for (int i = 0; i < n; i++) {
        graph* p = Graph[i];
        while (p) {
            addEdgeSort(head, i, p->wt, p->data);
            p = p->next;
        }
    }
}

// Function to display the edges in the sorted list
void displayEdges(edge* head) {
    while (head) {
        printf("Source: %d, Destination: %d, Weight: %d -> ", head->src, head->des, head->wt);
        head = head->next;
    }
    printf("NULL\n");
}
```

```c
// Main function for menu-driven operations
int main() {
  int n, choice;
  printf("Enter the number of vertices: ");
  scanf("%d", &n);

  graph* Graph[n];
  for (int i = 0; i < n; i++) {
    Graph[i] = NULL;
  }

  edge* edgeList = NULL;
  printf("\nMenu:\n");
    printf("1. Read Graph\n");
    printf("2. Display Graph\n");
    printf("3. Sort Edges\n");
    printf("4. Display Edges\n");
    printf("5. Exit\n");
  while (1) {

    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        readGraph(Graph, n);
```

```
            break;
        case 2:
            displayGraph(Graph, n);
            break;
        case 3:
            edgeList = NULL; // Reset edge list before sorting
            addGraphEdges(Graph, n, &edgeList);
            break;
        case 4:
            displayEdges(edgeList);
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid choice! Please enter a valid option.\n");
      }
    }
    return 0;
}
```

Output:

PS D:\c++\raj basnet> gcc q8.c

PS D:\c++\raj basnet> ./a.exe

Enter the number of vertices: 3

Menu:

1. Read Graph

2. Display Graph

3. Sort Edges

4. Display Edges

5. Exit

Enter your choice: 1

Enter the number of edges adjacent to vertex 0: 2

Enter the adjacent vertex: 1

Enter its weight: 2

Enter the adjacent vertex: 3

Enter its weight: 5

Enter the number of edges adjacent to vertex 1: 3

Enter the adjacent vertex: 1

Enter its weight: 2

Enter the adjacent vertex: 1

Enter its weight: 4

Enter the adjacent vertex: 2

Enter its weight: 5

Enter the number of edges adjacent to vertex 2: 2

Enter the adjacent vertex: 1

Enter its weight: 2

Enter the adjacent vertex: 3

Enter its weight: 2

Enter your choice: 2

Vertex 0: 1 (2) -> 3 (5) -> NULL

Vertex 1: 1 (2) -> 1 (4) -> 2 (5) -> NULL

Vertex 2: 1 (2) -> 3 (2) -> NULL

Enter your choice: 3

Enter your choice: 4

Source: 0, Destination: 1, Weight: 2 -> Source: 2, Destination: 3, Weight: 2 -> Source: 2, Destination: 1, Weight: 2 -> Source: 1, Destination: 1, Weight: 2 -> Source: 1, Destination: 1, Weight: 4 -> Source: 1, Destination: 2, Weight: 5 -> Source: 0, Destination: 3, Weight: 5 -> NULL

Enter your choice: 5

/*Q9. Write a menu driven program to implement DFS.

Name: Raj Basnet

Section : A2 Rollno.: 49

Course : B.Tech*/

Source Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


// Definition for graph node

typedef struct node {

  int data;

  struct node* next;

} graph;


// Function to create a new graph node

graph* create_list(int data) {

  graph* temp = (graph*)malloc(sizeof(graph));

  temp->data = data;

  temp->next = NULL;

  return temp;

}


// Function to insert a node into the graph

void insert(graph** head, int data) {
```

```
    graph* temp = create_list(data);
    if (*head == NULL) {
        *head = temp;
        return;
    }
    graph* temp2 = *head;
    while (temp2->next) {
        temp2 = temp2->next;
    }
    temp2->next = temp;
}

// Function to read the graph
void readGraph(graph* Graph[], int n) {
    for (int i = 0; i < n; i++) {
        int k;
        printf("Enter the number of edges adjacent to vertex %d: ", i);
        scanf("%d", &k);
        for (int j = 0; j < k; j++) {
            int y;
            printf("Enter the adjacent vertex: ");
            scanf("%d", &y);
            insert(&Graph[i], y);
        }
    }
}
```

```
// Function to display the graph
void displayGraph(graph* Graph[], int n) {
  for (int i = 0; i < n; i++) {
    printf("Vertex %d: ", i);
    graph* temp = Graph[i];
    while (temp) {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
    printf("NULL\n");
  }
}

// Function to perform Depth-First Search (DFS)
void dfs(int i, graph *Graph[], int vis[]) {
  vis[i] = 1;
  printf("%d ", i);
  graph *temp = Graph[i];
  while (temp) {
    if (!vis[temp->data]) {
      dfs(temp->data, Graph, vis);
    }
    temp = temp->next;
  }
}
```

```
// Main function for menu-driven operations
int main() {
  int n, choice;
  printf("Enter the number of vertices: ");
  scanf("%d", &n);

  graph* Graph[n];
  for (int i = 0; i < n; i++) {
    Graph[i] = NULL;
  }
  int vis[n];

  printf("\nMenu:\n");
  printf("1. Read Graph\n");
  printf("2. Display Graph\n");
  printf("3. Perform DFS\n");
  printf("4. Exit\n");

  while (1) {
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        readGraph(Graph, n);
```

```
        break;
    case 2:
        displayGraph(Graph, n);
        break;
    case 3:
        {
            int source;
            printf("Enter the source vertex for DFS: ");
            scanf("%d", &source);
            if (source < 0 || source >= n) {
                printf("Invalid source vertex! Please enter a valid vertex.\n");
            } else {
                // Reset the visited array before performing DFS
                for (int i = 0; i < n; i++) {
                    vis[i] = 0;
                }
                dfs(source, Graph, vis);
                printf("\n");
            }
        }
        break;
    case 4:
        exit(0);
    default:
        printf("Invalid choice! Please enter a valid option.\n");
}
```

```
    }
    return 0;
}
```

Output:

PS D:\c++\raj basnet> gcc q9.c

PS D:\c++\raj basnet> ./a.exe

Enter the number of vertices: 3

Menu:

1. Read Graph

2. Display Graph

3. Perform DFS

4. Exit

Enter your choice: 1

Enter the number of edges adjacent to vertex 0: 1

Enter the adjacent vertex: 2

Enter the number of edges adjacent to vertex 1: 1

Enter the adjacent vertex: 2

Enter the number of edges adjacent to vertex 2: 1

Enter the adjacent vertex: 0

Enter your choice: 2

Vertex 0: 2 -> NULL

Vertex 1: 2 -> NULL

Vertex 2: 0 -> NULL

Enter your choice: 3

Enter the source vertex for DFS: 0

0 2

Enter your choice: 4

/*Q10. Write a menu driven program to implement BFS.

Name: Raj Basnet

Section: A2 Rollno,: 49

Course : B.Tech */

Source Code:

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

#define MAX 100

// Definition for graph node
typedef struct node {
   int data;
   struct node* next;
} graph;

// Function to create a new graph node
graph* create_list(int data) {
   graph* temp = (graph*)malloc(sizeof(graph));
   temp->data = data;
   temp->next = NULL;
   return temp;
}

// Definition for the queue structure
```

```c
typedef struct queue {
   int* arr;
   int front;
   int rear;
   int size;
} queue;

// Function to create a new queue
queue* create_queue() {
   queue* temp = (queue*)malloc(sizeof(queue));
   temp->front = -1;
   temp->rear = -1;
   temp->size = MAX;
   temp->arr = (int*)malloc(sizeof(int) * MAX);
   return temp;
}

// Function to enqueue an element into the queue
void enqueue(queue* q, int data) {
   if (q->rear == q->size - 1) {
      printf("Queue is full!\n");
      return;
   }
   if (q->front == -1) {
      q->front = 0;
   }
```

```
    q->arr[++(q->rear)] = data;
}

// Function to dequeue an element from the queue
int dequeue(queue* q) {
    if (q->front == -1 || q->front > q->rear) {
        printf("Queue is empty!\n");
        return -1;
    }
    int data = q->arr[q->front++];
    if (q->front > q->rear) {
        q->front = q->rear = -1; // Reset queue
    }
    return data;
}

// Function to check if the queue is empty
bool is_empty(queue* q) {
    return q->front == -1;
}

// Function to insert a node into the graph
void insert(graph** head, int data) {
    graph* temp = create_list(data);
    if (*head == NULL) {
        *head = temp;
```

```
        return;
    }
    graph* temp2 = *head;
    while (temp2->next) {
        temp2 = temp2->next;
    }
    temp2->next = temp;
}

// Function to read the graph
void readGraph(graph* Graph[], int n) {
    for (int i = 0; i < n; i++) {
        int k;
        printf("Enter the number of edges adjacent to vertex %d: ", i);
        scanf("%d", &k);
        for (int j = 0; j < k; j++) {
            int y;
            printf("Enter the adjacent vertex: ");
            scanf("%d", &y);
            insert(&Graph[i], y);
        }
    }
}

// Function to display the graph
void displayGraph(graph* Graph[], int n) {
```

```
    for (int i = 0; i < n; i++) {
        printf("Vertex %d: ", i);
        graph* temp = Graph[i];
        while (temp) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

// Function to perform Breadth-First Search (BFS)
void bfs(graph* Graph[], int n) {
    queue* q = create_queue();
    enqueue(q, 0);
    int vis[n];
    for (int i = 0; i < n; i++) {
        vis[i] = 0;
    }
    vis[0] = 1;

    while (!is_empty(q)) {
        int node = dequeue(q);
        printf("%d ", node);
        graph* temp = Graph[node];
        while (temp) {
```

```
            if (!vis[temp->data]) {
                enqueue(q, temp->data);
                vis[temp->data] = 1;
            }
            temp = temp->next;
        }
    }
    printf("\n");
}

// Main function for menu-driven operations
int main() {
    int n, choice;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    graph* Graph[n];
    for (int i = 0; i < n; i++) {
        Graph[i] = NULL;
    }
        printf("\nMenu:\n");
        printf("1. Read Graph\n");
        printf("2. Display Graph\n");
        printf("3. Perform BFS\n");
        printf("4. Exit\n");
```

```c
  while (1) {
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        readGraph(Graph, n);
        break;
      case 2:
        displayGraph(Graph, n);
        break;
      case 3:
        bfs(Graph, n);
        break;
      case 4:
        exit(0);
      default:
        printf("Invalid choice! Please enter a valid option.\n");
    }
  }
  return 0;
}
```

OUTPUT:

PS D:\c++\raj basnet> gcc q10.c

PS D:\c++\raj basnet> ./a.exe

Enter the number of vertices: 3

Menu:

1. Read Graph

2. Display Graph

3. Perform BFS

4. Exit

Enter your choice: 1

Enter the number of edges adjacent to vertex 0: 1

Enter the adjacent vertex: 2

Enter the number of edges adjacent to vertex 1: 2

Enter the adjacent vertex: 0

Enter the adjacent vertex: 2

Enter the number of edges adjacent to vertex 2: 1

Enter the adjacent vertex: 0

Enter your choice: 2

Vertex 0: 2 -> NULL

Vertex 1: 0 -> 2 -> NULL

Vertex 2: 0 -> NULL

Enter your choice: 3

0 2

Enter your choice: 4