1. Graph Using an Adjacency Matrix
Functionality:
• Accept vertices and edges and store in an adjacency matrix.
• Display the adjacency matrix.
• Compute and print the indegree for all vertices.

```c
// File: Graph_AdjMatrix_Indegree.c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int vertices, edges;
    printf("Enter number of vertices: ");
    scanf("%d", &vertices);
    printf("Enter number of edges: ");
    scanf("%d", &edges);

    // Allocate adjacency matrix dynamically.
    int **adjMatrix = malloc(vertices * sizeof(int *));
    for (int i = 0; i < vertices; i++) {
        adjMatrix[i] = calloc(vertices, sizeof(int));
    }

    printf("Enter each edge (u v) [vertex numbering starts at 0]:\n");
    for (int i = 0; i < edges; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        if(u >= 0 && u < vertices && v >= 0 && v < vertices)
            adjMatrix[u][v] = 1;
    }

    // Display the matrix.
    printf("\nAdjacency Matrix:\n");
    for (int i = 0; i < vertices; i++){
        for (int j = 0; j < vertices; j++){
            printf("%d ", adjMatrix[i][j]);
        }
        printf("\n");
    }

    // Compute and print the indegree for each vertex.
    printf("\nIndegree of each vertex:\n");
    for (int j = 0; j < vertices; j++){
        int indegree = 0;
        for (int i = 0; i < vertices; i++){
            if(adjMatrix[i][j] == 1)
                indegree++;
        }
        printf("Vertex %d: %d\n", j, indegree);
    }

    // Free the allocated memory.
    for (int i = 0; i < vertices; i++){
        free(adjMatrix[i]);
    }
    free(adjMatrix);

    return 0;
}
```

---

2. Graph Traversal Using BFS (Adjacency Matrix)
Functionality:
- Create the adjacency matrix as before.
- Traverse the graph using Breadth First Search (BFS).

```c
// File: Graph_AdjMatrix_BFS.c
#include <stdio.h>
#include <stdlib.h>
#define MAX_QUEUE 100

// Simple queue implementation.
typedef struct {
    int items[MAX_QUEUE];
    int front, rear;
} Queue;

void initQueue(Queue *q) {
    q->front = 0;
    q->rear = -1;
}

int isEmpty(Queue *q) {
    return q->rear < q->front;
}

void enqueue(Queue *q, int value) {
    if(q->rear < MAX_QUEUE - 1)
        q->items[++q->rear] = value;
}

int dequeue(Queue *q) {
    if(!isEmpty(q))
        return q->items[q->front++];
    return -1;
}

int main() {
    int vertices, edges;
    printf("Enter number of vertices: ");
    scanf("%d", &vertices);
    printf("Enter number of edges: ");
    scanf("%d", &edges);

    // Build the adjacency matrix.
    int **matrix = malloc(vertices * sizeof(int *));
    for (int i = 0; i < vertices; i++)
        matrix[i] = calloc(vertices, sizeof(int));

    printf("Enter each edge (u v) [0-indexed]:\n");
    for (int i = 0; i < edges; i++){
        int u, v;
        scanf("%d %d", &u, &v);
        if(u >= 0 && u < vertices && v >= 0 && v < vertices)
            matrix[u][v] = 1;
    }

    int start;
    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &start);

    int *visited = calloc(vertices, sizeof(int));
    Queue q;
    initQueue(&q);
    visited[start] = 1;
    enqueue(&q, start);

    printf("\nBFS Traversal starting from vertex %d: ", start);
    while(!isEmpty(&q)) {
        int curr = dequeue(&q);
        printf("%d ", curr);
        for (int i = 0; i < vertices; i++) {
            if(matrix[curr][i] && !visited[i]) {
                visited[i] = 1;
                enqueue(&q, i);
            }
        }
    }
    printf("\n");

    free(visited);
    for (int i = 0; i < vertices; i++)
        free(matrix[i]);
    free(matrix);

    return 0;
}
```

---

3. Graph Traversal Using DFS (Adjacency Matrix)

Functionality:

- Create and populate the adjacency matrix.
- Traverse the graph using recursive Depth First Search (DFS).

```c
// File: Graph_AdjMatrix_DFS.c
#include <stdio.h>
#include <stdlib.h>

void DFS(int **matrix, int vertices, int vertex, int *visited) {
    visited[vertex] = 1;
    printf("%d ", vertex);
    for (int i = 0; i < vertices; i++) {
        if(matrix[vertex][i] && !visited[i])
            DFS(matrix, vertices, i, visited);
    }
}

int main() {
    int vertices, edges;
    printf("Enter number of vertices: ");
    scanf("%d", &vertices);
    printf("Enter number of edges: ");
    scanf("%d", &edges);

    // Build the adjacency matrix.
    int **matrix = malloc(vertices * sizeof(int *));
    for (int i = 0; i < vertices; i++)
        matrix[i] = calloc(vertices, sizeof(int));

    printf("Enter each edge (u v) [0-indexed]:\n");
    for (int i = 0; i < edges; i++){
        int u, v;
        scanf("%d %d", &u, &v);
        if(u >= 0 && u < vertices && v >= 0 && v < vertices)
```

```
            matrix[u][v] = 1;
    }

    int start;
    printf("Enter starting vertex for DFS: ");
    scanf("%d", &start);

    int *visited = calloc(vertices, sizeof(int));
    printf("\nDFS Traversal starting from vertex %d: ", start);
    DFS(matrix, vertices, start, visited);
    printf("\n");

    free(visited);
    for (int i = 0; i < vertices; i++)
        free(matrix[i]);
    free(matrix);

    return 0;
}
```

---

4. Graph Using an Adjacency List

Functionality:

- Accept vertices and edges and store the graph using linked lists.
- Display the complete adjacency list.

```c
// File: Graph_AdjList.c
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int vertex;
    struct Node *next;
} Node;

Node* createNode(int vertex) {
    Node *newNode = malloc(sizeof(Node));
    newNode->vertex = vertex;
    newNode->next = NULL;
    return newNode;
}

void addEdge(Node **adjList, int u, int v) {
    Node *newNode = createNode(v);
    newNode->next = adjList[u];
    adjList[u] = newNode;
}

void displayGraph(Node **adjList, int vertices) {
    for (int i = 0; i < vertices; i++){
        printf("Vertex %d: ", i);
        Node *temp = adjList[i];
        while(temp != NULL) {
            printf("-> %d ", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }
}

int main() {
    int vertices, edges;
    printf("Enter number of vertices: ");
    scanf("%d", &vertices);
    printf("Enter number of edges: ");
    scanf("%d", &edges);

    // Create an array of linked-list pointers.
    Node **adjList = malloc(vertices * sizeof(Node *));
    for (int i = 0; i < vertices; i++)
        adjList[i] = NULL;

    printf("Enter each edge (u v) [0-indexed]:\n");
    for (int i = 0; i < edges; i++){
        int u, v;
        scanf("%d %d", &u, &v);
        if(u >= 0 && u < vertices && v >= 0 && v < vertices)
            addEdge(adjList, u, v);
    }

    printf("\nAdjacency List:\n");
    displayGraph(adjList, vertices);

    // Free allocated memory.
    for (int i = 0; i < vertices; i++){
        Node *temp = adjList[i];
        while(temp) {
            Node *t = temp;
            temp = temp->next;
            free(t);
        }
    }
    free(adjList);

    return 0;
}
```

BST Problems

1. BST – Create, Insert & Inorder Traversal

Functionality:

• Use a BST library that provides a function to create a new node and insert nodes into the BST.
• Traverse the tree using inorder traversal.

```c
// File: BST_Inorder.c
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node *left, *right;
} Node;

Node* createNode(int data) {
    Node *newNode = malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}
```

```c
Node* insert(Node *root, int data) {
    if(root == NULL)
        return createNode(data);
    if(data < root->data)
        root->left = insert(root->left, data);
    else
        root->right = insert(root->right, data);
    return root;
}

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

int main() {
    Node *root = NULL;
    int choice, data;

    while(1) {
        printf("\nBST Menu (Inorder Traversal):\n");
        printf("1. Insert Node\n");
        printf("2. Display Inorder Traversal\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if(choice == 1) {
            printf("Enter number to insert: ");
            scanf("%d", &data);
            root = insert(root, data);
        } else if(choice == 2) {
            printf("Inorder Traversal: ");
            inorder(root);
            printf("\n");
        } else if(choice == 3) {
            break;
        } else {
            printf("Invalid choice.\n");
        }
    }
    return 0;
}
```

---

2. BST – Create, Search & Preorder Traversal
Functionality:
- Insert nodes into a BST.
- Traverse using preorder and search for a specified key.

```c
// File: BST_Preorder_Search.c
#include <stdio.h>
#include <stdlib.h>

typedef struct BSTNode {
    int data;
    struct BSTNode *left, *right;
} BSTNode;
```

```
BSTNode* createNode(int data) {
    BSTNode* newNode = malloc(sizeof(BSTNode));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

BSTNode* insert(BSTNode *root, int data) {
    if(root == NULL)
        return createNode(data);
    if(data < root->data)
        root->left = insert(root->left, data);
    else
        root->right = insert(root->right, data);
    return root;
}

void preorder(BSTNode *root) {
    if(root) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

BSTNode* search(BSTNode *root, int key) {
    if(root == NULL || root->data == key)
        return root;
    if(key < root->data)
        return search(root->left, key);
    else
        return search(root->right, key);
}

int main() {
    BSTNode *root = NULL;
    int choice, data, key;
    BSTNode *found = NULL;

    while(1) {
        printf("\nBST Menu (Preorder & Search):\n");
        printf("1. Insert Node\n");
        printf("2. Preorder Traversal\n");
        printf("3. Search for a Key\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if(choice == 1) {
            printf("Enter number to insert: ");
            scanf("%d", &data);
            root = insert(root, data);
        } else if(choice == 2) {
            printf("Preorder Traversal: ");
            preorder(root);
            printf("\n");
        } else if(choice == 3) {
            printf("Enter key to search: ");
            scanf("%d", &key);
            found = search(root, key);
            if(found)
                printf("Key %d found in the BST.\n", key);
            else
```

```
                printf("Key %d not found in the BST.\n", key);
        } else if(choice == 4) {
            break;
        } else {
            printf("Invalid choice.\n");
        }
    }
    return 0;
}
```

---

3. BST – Create, Insert & Postorder Traversal

Functionality:

• Insert nodes into a BST and then display a postorder traversal.

```c
// File: BST_Postorder.c
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node *left, *right;
} Node;

Node* createNode(int data) {
    Node* newNode = malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

Node* insert(Node *root, int data) {
    if(root == NULL)
        return createNode(data);
    if(data < root->data)
        root->left = insert(root->left, data);
    else
        root->right = insert(root->right, data);
    return root;
}

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

int main() {
    Node *root = NULL;
    int choice, data;

    while(1) {
        printf("\nBST Menu (Postorder Traversal):\n");
        printf("1. Insert Node\n");
        printf("2. Display Postorder Traversal\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        if(choice == 1) {
            printf("Enter number to insert: ");
            scanf("%d", &data);
            root = insert(root, data);
        } else if(choice == 2) {
            printf("Postorder Traversal: ");
            postorder(root);
            printf("\n");
        } else if(choice == 3) {
            break;
        } else {
            printf("Invalid choice.\n");
        }
    }
    return 0;
}
```

---

4. BST – Count Leaf Nodes

Functionality:

- Build a BST and count the number of leaf nodes.

```c
// File: BST_CountLeafNodes.c
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node *left, *right;
} Node;

Node* createNode (int data) {
    Node* newNode = malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

Node* insert(Node *root, int data) {
    if(root == NULL)
        return createNode(data);
    if(data < root->data)
        root->left = insert(root->left, data);
    else
        root->right = insert(root->right, data);
    return root;
}

int countLeafNodes(Node *root) {
    if(root == NULL)
        return 0;
    if(root->left == NULL && root->right == NULL)
        return 1;
    return countLeafNodes(root->left) + countLeafNodes(root->right);
}

int main() {
    Node *root = NULL;
    int choice, data;

    while(1) {
        printf("\nBST Menu (Count Leaf Nodes):\n");
        printf("1. Insert Node\n");
        printf("2. Count Leaf Nodes\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if(choice == 1) {
            printf("Enter number to insert: ");
            scanf("%d", &data);
            root = insert(root, data);
        } else if(choice == 2) {
            printf("Total leaf nodes: %d\n", countLeafNodes(root));
        } else if(choice == 3) {
            break;
        } else {
            printf("Invalid choice.\n");
        }
    }
    return 0;
}
```

---

5. BST – Count Total Nodes

Functionality:

- Insert nodes into the BST and count the total number of nodes.

```c
// File: BST_CountTotalNodes.c
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node *left, *right;
} Node;

Node* createNode (int data) {
    Node* newNode = malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

Node* insert(Node *root, int data) {
    if(root == NULL)
        return createNode(data);
    if(data < root->data)
        root->left = insert(root->left, data);
    else
        root->right = insert(root->right, data);
    return root;
}

int countNodes(Node *root) {
    if(root == NULL)
        return 0;
    return 1 + countNodes(root->left) + countNodes(root->right);
}

int main() {
    Node *root = NULL;
```

~~~~
    int choice, data;

    while(1) {
        printf("\nBST Menu (Count Total Nodes):\n");
        printf("1. Insert Node\n");
        printf("2. Count Total Nodes\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if(choice == 1) {
            printf("Enter number to insert: ");
            scanf("%d", &data);
            root = insert(root, data);
        } else if(choice == 2) {
            printf("Total nodes in BST: %d\n", countNodes(root));
        } else if(choice == 3) {
            break;
        } else {
            printf("Invalid choice.\n");
        }
    }
    return 0;
}
```
~~~~