**Assignment 7**

**Name**: Komal Potdar

**Roll No**: 92

**PRN No**: 12320165

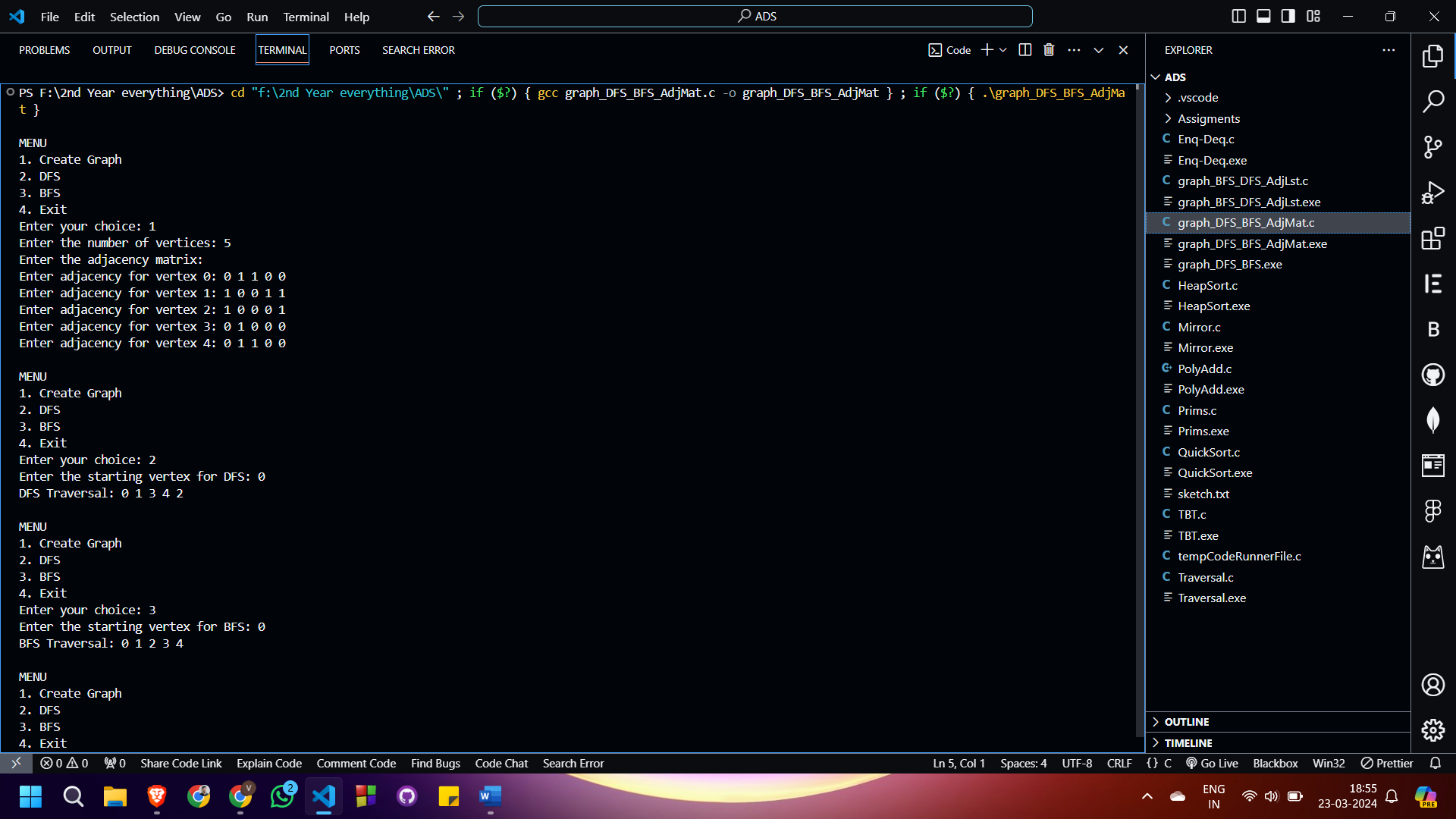
**Div**: CS B SY

**Batch**: 3

1. **Write a program in C to traverse DFS and BFS on graph using Adjcency matrix.**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #define MAX\_VERTICES 20  int adjacencyMatrix[MAX\_VERTICES][MAX\_VERTICES];  int visited[MAX\_VERTICES];  int queue[MAX\_VERTICES];  int front = -1, rear = -1;  int vertices;  void createGraph()  {      int i, j;      printf("Enter the number of vertices: ");      scanf("%d", &vertices);      printf("Enter the adjacency matrix:\n");      for (i = 0; i < vertices; i++)      {          printf("Enter adjacency for vertex %d: ", i);          for (j = 0; j < vertices; j++)          {              scanf("%d", &adjacencyMatrix[i][j]);          }      }  }  void initializeVisited()  {      int i;      for (i = 0; i < vertices; i++)      {          visited[i] = 0;      }  }  void enqueue(int vertex)  {      if (rear == MAX\_VERTICES - 1)      {          printf("Queue Overflow!\n");      }      else      {          if (front == -1)          {              front = 0;          }          rear++;          queue[rear] = vertex;      }  }  int dequeue()  {      int vertex;      if (front == -1 || front > rear)      {          printf("Queue Underflow!\n");          return -1;      }      else      {          vertex = queue[front];          front++;          return vertex;      }  }  int isQueueEmpty()  {      if (front == -1 || front > rear)      {          return 1;      }      else      {          return 0;      }  }  void DFS(int vertex)  {      int i;      printf("%d ", vertex);      visited[vertex] = 1;      for (i = 0; i < vertices; i++)      {          if (adjacencyMatrix[vertex][i] == 1 && !visited[i])          {              DFS(i);          }      }  }  void BFS(int vertex)  {      int i, dequeueVertex;      printf("%d ", vertex);      visited[vertex] = 1;      enqueue(vertex);      while (!isQueueEmpty())      {          dequeueVertex = dequeue();          for (i = 0; i < vertices; i++)          {              if (adjacencyMatrix[dequeueVertex][i] == 1 && !visited[i])              {                  printf("%d ", i);                  visited[i] = 1;                  enqueue(i);              }          }      }  }  int main()  {      int choice, startVertex;      while (1)      {          printf("\nMENU\n");          printf("1. Create Graph\n");          printf("2. DFS\n");          printf("3. BFS\n");          printf("4. Exit\n");          printf("Enter your choice: ");          scanf("%d", &choice);          switch (choice)          {          case 1:              createGraph();              break;          case 2:              initializeVisited();              printf("Enter the starting vertex for DFS: ");              scanf("%d", &startVertex);              printf("DFS Traversal: ");              DFS(startVertex);              printf("\n");              break;          case 3:              initializeVisited();              printf("Enter the starting vertex for BFS: ");              scanf("%d", &startVertex);              printf("BFS Traversal: ");              BFS(startVertex);              printf("\n");              break;          case 4:              printf("Exiting program...\n");              exit(0);          default:              printf("Invalid choice! Please enter a valid option.\n");          }      }      return 0;  } |

Output:



1. **Write a program in C to traverse DFS and BFS on graph using Adjcency List.**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  struct AdjListNode {      int dest;      struct AdjListNode\* next;  };  struct AdjList {      struct AdjListNode\* head;  };  struct Graph {      int V;      struct AdjList\* array;  };  struct AdjListNode\* newAdjListNode(int dest) {      struct AdjListNode\* newNode = (struct AdjListNode\*)malloc(sizeof(struct AdjListNode));      newNode->dest = dest;      newNode->next = NULL;      return newNode;  }  struct Graph\* createGraph(int V) {      struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));      graph->V = V;      graph->array = (struct AdjList\*)malloc(V \* sizeof(struct AdjList));      for (int i = 0; i < V; ++i)          graph->array[i].head = NULL;      return graph;  }  void addEdge(struct Graph\* graph, int src, int dest) {      struct AdjListNode\* newNode = newAdjListNode(dest);      newNode->next = graph->array[src].head;      graph->array[src].head = newNode;        newNode = newAdjListNode(src);      newNode->next = graph->array[dest].head;      graph->array[dest].head = newNode;  }  void DFS(struct Graph\* graph, int start) {      int\* visited = (int\*)calloc(graph->V, sizeof(int));      int stack[graph->V];      int top = -1;      visited[start] = 1;      stack[++top] = start;      while (top != -1) {          int current = stack[top--];          printf("%d ", current);          struct AdjListNode\* temp = graph->array[current].head;          while (temp != NULL) {              if (!visited[temp->dest]) {                  visited[temp->dest] = 1;                  stack[++top] = temp->dest;              }              temp = temp->next;          }      }      free(visited);  }  void BFS(struct Graph\* graph, int start) {      int\* visited = (int\*)calloc(graph->V, sizeof(int));      int queue[graph->V];      int front = 0, rear = 0;      visited[start] = 1;      queue[rear++] = start;      while (front < rear) {          int current = queue[front++];          printf("%d ", current);          struct AdjListNode\* temp = graph->array[current].head;          while (temp != NULL) {              if (!visited[temp->dest]) {                  visited[temp->dest] = 1;                  queue[rear++] = temp->dest;              }              temp = temp->next;          }      }      free(visited);  }  void displayMenu() {      printf("\nMenu:\n");      printf("1. Add Edge\n");      printf("2. Depth First Search (DFS)\n");      printf("3. Breadth First Search (BFS)\n");      printf("4. Exit\n");  }  int main() {      int V, choice, src, dest;      printf("Enter the number of vertices: ");      scanf("%d", &V);      struct Graph\* graph = createGraph(V);      while (1) {          displayMenu();          printf("Enter your choice: ");          scanf("%d", &choice);          switch (choice) {              case 1:                  printf("Enter source and destination vertices (0 to %d): ", V - 1);                  scanf("%d %d", &src, &dest);                  if (src >= 0 && src < V && dest >= 0 && dest < V)                      addEdge(graph, src, dest);                  else                      printf("Invalid vertices! Please enter vertices within the range.\n");                  break;              case 2:                  printf("DFS Traversal: ");                  DFS(graph, 0);                  printf("\n");                  break;              case 3:                  printf("BFS Traversal: ");                  BFS(graph, 0);                  printf("\n");                  break;              case 4:                  printf("Exiting...\n");                  exit(0);              default:                  printf("Invalid choice! Please enter a valid option.\n");          }      }      return 0;  } |

Output:

