Week-10 UE20CS207 DSLAB

- Name : P K Navin Shrinivas
- SRN : PES2UG20CS237
- Section : D
- Batch : 2

HashTable DFS BFS

Code :

main.c

```
#include "1_1.h"
#include <stdio.h>

int main() {
  // little queue stuff
  int queue[QUEUESIZE];
  int top = -1, front = -1;
  //------------------
  int a[graphvertices][graphvertices] = {0}; // initializir
  printf(" This program considers all edges undirected\n \r
  while (true) {
    printf("1.Inserted edge to graph \n");
    printf("2.DFS using Recursion \n");
    printf("3.BFS using Queue \n");
    printf("4.Indegree of a vertice \n");
    printf("5.Out degree of a vertice \n");
    printf("6.Find all paths between two vertices \n");
    printf("7.Are two vertices connected? \n");
    printf("Enter choice : ");
    int choice;
    scanf("%d", &choice);
    if (choice == 1) {
      int m, n;
      printf(" \n Enter two vertices the edge connects : ")
      scanf("%d %d", &m, &n);
      GraphInsertEdge(a, n, m);
    } else if (choice == 2) {
      int j;
      printf("Enter node to start traversal : ");
      scanf("%d", &j);
      printf("\n ");
      bool visited[graphvertices] = {false};
      GraphDFS(a, visited, j);
      printf("End of graph traversal \n \n");
    } else if (choice == 3) {
      int j;
      printf("Enter node to start traversal : ");
      scanf("%d", &j);
      printf("\n ");
      bool visited[graphvertices] = {false};
      visited[j] = true;
      queuepush(queue, &top, &front, j);
      GraphBFS(a, j, queue, &top, &front, visited);
      printf(" End of graph traversal\n \n");
    } else if (choice == 4) {
      int v;
      printf("Enter vertice to find Indegree :  ");
      scanf("%d", &v);
```

```
      printf(" \n Indegree of %d : %d \n \n", v, GraphIndeg
    } else if (choice == 5) {
      int v;
      printf("Enter vertice to find Outdegree :  ");
      scanf("%d", &v);
      printf(" \n Outdegree of %d : %d \n \n", v, GraphOutd
    } else if (choice == 6) {
      int s, d;
      printf("Enter source and destination with space : ");
      scanf("%d %d", &s, &d);
      printf("\n All paths between the two vertices : \n\n"
      bool visited[graphvertices] = {false};
      int path[graphvertices] = {0};
      GraphAllPath(a, visited, path, s, d);
    } else if (choice == 7) {
      int c, d;
      printf("Enter two vertices : ");
      scanf("%d %d", &c, &d);
      IsConnected(a, c, d);
    } else {
      printf("Buh Byeee :) \n");
      return 0;
    }
  }
}
```

1_1.h

```c
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<string.h>
#define graphvertices 1000

void GraphInsertEdge(int a[][graphvertices], int n, int m);
void GraphDFS(int a[][graphvertices], bool visited[graphver
void GraphBFS(int a[][graphvertices], int j, int *queue, in

//queue stuff
#define QUEUESIZE 100

void queuepush(int* queue , int* top , int* front,int e);
void queuepop(int* queue,int* top , int* front);
void queuepeek(int* queue , int* top , int* front);
void queuedisplay(int* queue , int* top, int* front);
int GraphIndegree(int a[][graphvertices], int v);
int GraphOutdegree(int a[][graphvertices], int v);
void GraphAllPath(int a[][graphvertices], bool visited[grap
void IsConnected(int a[][graphvertices], int c, int d);
```

1_1.c

```
#include "1_1.h"

void queuepush(int *queue, int *top, int *front, int e) {
  if (*top == QUEUESIZE - 1) {
    return;
  } else if (*top == -1 && *front == -1) {
    *top = 0;
    *front = 0;
    *(queue + *top) = e;
    return;
  } else {
    *top = *top + 1;
    *(queue + *top) = e;
    return;
  }
}

void queuepop(int *queue, int *top, int *front) {
  if (*top == -1 && *front == -1) {
    return;
  } else if (*front == *top) {
    printf("%d", *(queue + *front));
    *top = -1;
    *front = -1;
    return;
  } else {
    printf("%d", *(queue + *front));
    *front = *front + 1;
  }
}

int QueuePeek(int *queue, int *top, int *front) {
  if (*top == -1 && *front == -1)
    return -1;
  else
    return *(queue + *front);
}
void GraphInsertEdge(int a[][graphvertices], int n, int m)
  if (a[n][m] == 1 || a[m][n] == 1) {
    printf("\n This edge already exists! \n \n");
    return;
  } else {
    a[n][m] = 1;
    a[m][n] = 1;
    printf("\n Edge inserted to graph \n \n");
    return;
  }
}
```

```
void GraphDFS(int a[][graphvertices], bool visited[graphver
  if (visited[j] == true) {
    return;
  } else {
    printf("%d ->", j);
    visited[j] = true;
    for (int i = 0; i < graphvertices; i++) {
      if (a[j][i] == 1) {
        GraphDFS(a, visited, i);
      }
    }
  }
}

void GraphAllPath(int a[][graphvertices], bool visited[grap
                  int *path, int s, int d) {
  if (visited[s] == true) {
    return;
  } else if (s == d) {
    for (int i = 0; i < graphvertices; i++) {
      if (path[i] != 0)
        printf(" %d->", path[i]);
    }
    printf("%d\n", d);
  } else {
    visited[s] = true;
    for (int j = 0; j < graphvertices; j++) {
      if (path[j] == 0) {
        path[j] = s;
        break;
      }
    }
    for (int i = 0; i < graphvertices; i++) {
      if (a[s][i] == 1) {
        GraphAllPath(a, visited, path, i, d);
      }
    }
  }

  // after done with this vertex , i.e after finding one pa
  // and prepare for second path big brain smortness
  visited[s] = false;
  int j;
  for (int j = 0; j < graphvertices; j++) {
    if (path[j] == s) {
      path[j] = 0;
      break;
```

```
    }
  }
}

void GraphBFS(int a[][graphvertices], int j, int *queue, ir
              bool visited[graphvertices]) {
  while (*top != -1 && *front != -1) {
    int s = QueuePeek(queue, top, front);
    queuepop(queue, top, front);
    printf(" ->");
    for (int i = 0; i < graphvertices; i++) {
      if (a[s][i] == 1) {
        if (visited[i] == true)
          continue;
        else {
          queuepush(queue, top, front, i);
          visited[i] = true;
        }
      }
    }
  }
}

int GraphIndegree(int a[][graphvertices], int v) {
  // every edge that point to the vertex, I know this is ur
  // both In and Out degree will be the same , but eh!
  int count = 0;
  for (int i = 0; i < graphvertices; i++) {
    if (a[i][v] == 1)
      count++;
  }
  return count;
}

int GraphOutdegree(int a[][graphvertices], int v) {
  // every edge that point to the vertex, I know this is ur
  // both In and Out degree will be the same , but eh!
  int count = 0;
  for (int i = 0; i < graphvertices; i++) {
    if (a[v][i] == 1)
      count++;
  }
  return count;
}

void IsConnected(int a[][graphvertices], int c, int d) {
  bool flag = false;
  // First using BFS :
```

```
  bool visited[graphvertices] = {false};
  int queue[QUEUESIZE];
  int top = -1, front = -1;
  queuepush(queue, &top, &front, c);
  while (top != -1 && front != -1) {
    int s = QueuePeek(queue, &top, &front);
    queuepop(queue, &top, &front);
    printf(" ->");
    if (s == d) {
      flag = true;
      printf("\n Yup!The node are connected :) \n \n");
    }
    for (int i = 0; i < graphvertices; i++) {
      if (a[s][i] == 1) {
        if (visited[i] == true)
          continue;
        else {
          queuepush(queue, &top, &front, i);
          visited[i] = true;
        }
      }
    }
  }
  if (!flag) {
    printf("\n No , they are not connected :( \n \n");
    return;
  }
  return;

  // Using BFs : ahhh you get the idea , imma implement , l
}
```