#include <stdlib.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 5

struct Vertex {

char label;

bool visited;

};

//queue variables

int queue[MAX];

int rear = -1;

int front = 0;

int queueItemCount = 0;

//graph variables

//array of vertices

struct Vertex\* lstVertices[MAX];

//adjacency matrix

int adjMatrix[MAX][MAX];

//vertex count

int vertexCount = 0;

//queue functions

void insert(int data) {

queue[++rear] = data;

queueItemCount++;

}

int removeData() {

queueItemCount--;

return queue[front++];

}

bool isQueueEmpty() {

return queueItemCount == 0;

}

//graph functions

//add vertex to the vertex list

void addVertex(char label) {

struct Vertex\* vertex = (struct Vertex\*) malloc(sizeof(struct Vertex));

vertex->label = label;

vertex->visited = false;

lstVertices[vertexCount++] = vertex;

}

//add edge to edge array

void addEdge(int start,int end) {

adjMatrix[start][end] = 1;

adjMatrix[end][start] = 1;

}

//display the vertex

void displayVertex(int vertexIndex) {

printf("%c ",lstVertices[vertexIndex]->label);

}

//get the adjacent unvisited vertex

int getAdjUnvisitedVertex(int vertexIndex) {

int i;

for(i = 0; i<vertexCount; i++) {

if(adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited == false)

return i;

}

return -1;

}

void breadthFirstSearch() {

int i;

//mark first node as visited

lstVertices[0]->visited = true;

//display the vertex

displayVertex(0);

//insert vertex index in queue

insert(0);

int unvisitedVertex;

while(!isQueueEmpty()) {

//get the unvisited vertex of vertex which is at front of the queue

int tempVertex = removeData();

//no adjacent vertex found

while((unvisitedVertex = getAdjUnvisitedVertex(tempVertex)) != -1) {

lstVertices[unvisitedVertex]->visited = true;

displayVertex(unvisitedVertex);

insert(unvisitedVertex);

}

}

//queue is empty, search is complete, reset the visited flag

for(i = 0;i<vertexCount;i++) {

lstVertices[i]->visited = false;

}

}

int main() {

int i, j;

for(i = 0; i<MAX; i++) { // set adjacency

for(j = 0; j<MAX; j++) // matrix to 0

adjMatrix[i][j] = 0;

}

addVertex('S'); // 0

addVertex('A'); // 1

addVertex('B'); // 2

addVertex('C'); // 3

addVertex('D'); // 4

addEdge(0, 1); // S - A

addEdge(0, 2); // S - B

addEdge(0, 3); // S - C

addEdge(1, 4); // A - D

addEdge(2, 4); // B - D

addEdge(3, 4); // C - D

printf("\nBreadth First Search: ");

breadthFirstSearch();

return 0;

}

Output

Breadth First Search: S A B C D