Experiments\bfs.c

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

#define SIZE 40

// Queue structure
struct queue {
    int items[SIZE];
    int front;
    int rear;
};

// Node structure for adjacency list
struct node {
    int vertex;
    struct node* next;
};

// Graph structure
struct Graph {
    int numVertices;
    struct node** adjLists;
    int* visited;
};

// Function to create a node
struct node* createNode(int v) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    if (!newNode) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

// Function to create a graph
struct Graph* createGraph(int vertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    if (!graph) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    graph->numVertices = vertices;
    graph->adjLists = (struct node**)malloc(vertices * sizeof(struct node*));
    graph->visited = (int*)malloc(vertices * sizeof(int));

    for (int i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    return graph;
}

// Function to add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Add edge from dest to src (since the graph is undirected)
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// Function to create a queue
struct queue* createQueue() {
    struct queue* q = (struct queue*)malloc(sizeof(struct queue));
    if (!q) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    q->front = -1;
    q->rear = -1;
    return q;
}

// Function to check if the queue is empty
int isEmpty(struct queue* q) {
    return q->rear == -1;
}

// Function to add an element to the queue
void enqueue(struct queue* q, int value) {
    if (q->rear == SIZE - 1) {
        printf("\nQueue is full!\n");
    } else {
        if (q->front == -1)
            q->front = 0;
        q->rear++;
        q->items[q->rear] = value;
    }
}

```

```c
// Function to remove an element from the queue
int dequeue(struct queue* q) {
    int item = -1;
    if (isEmpty(q)) {
        printf("Queue is empty\n");
    } else {
        item = q->items[q->front];
        q->front++;
        if (q->front > q->rear) {
            q->front = q->rear = -1; // Reset queue
        }
    }
    return item;
}

// Function to print the queue
void printQueue(struct queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
    } else {
        printf("\nQueue contains: ");
        for (int i = q->front; i <= q->rear; i++) {
            printf("%d ", q->items[i]);
        }
        printf("\n");
    }
}

// BFS algorithm
void bfs(struct Graph* graph, int startVertex) {
    struct queue* q = createQueue();

    graph->visited[startVertex] = 1;
    enqueue(q, startVertex);

    while (!isEmpty(q)) {
        printQueue(q);
        int currentVertex = dequeue(q);
        printf("Visited %d\n", currentVertex);

        struct node* temp = graph->adjLists[currentVertex];

        while (temp) {
            int adjVertex = temp->vertex;

            if (graph->visited[adjVertex] == 0) {
                graph->visited[adjVertex] = 1;
                enqueue(q, adjVertex);
            }
            temp = temp->next;
        }
    }

    // Free the queue
    free(q);
}

// Main function
int main() {
    struct Graph* graph = createGraph(6);

    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 4);
    addEdge(graph, 1, 3);
    addEdge(graph, 2, 4);
    addEdge(graph, 3, 4);

    printf("BFS Traversal starting from vertex 0:\n");
    bfs(graph, 0);

    // Free allocated memory for the graph
    for (int i = 0; i < graph->numVertices; i++) {
        struct node* temp = graph->adjLists[i];
        while (temp) {
            struct node* toFree = temp;
            temp = temp->next;
            free(toFree);
        }
    }
    free(graph->adjLists);
    free(graph->visited);
    free(graph);

    return 0;
}
/*
Output:
PS C:\Users\gagan\Documents\Data_Structure_And_Algorithm\Experiments> cd "c:\Users\gagan\Documents\Data_Structure_And_Algorithm\Experiments\" ; if ($?) { gcc bfs.c -o bfs } ; if ($?) { .\bfs }
BFS Traversal starting from vertex 0:

Queue contains: 0
Visited 0

Queue contains: 2 1
Visited 2

```

```
Queue contains: 1 4
Visited 1

Queue contains: 4 3
Visited 4

Queue contains: 3
Visited 3
*/
```