10/18/24, 7:02 AM bfs.c

Experiments\bfs.c

```
#include <stdio.h>
 2
   #include <stdlib.h>
 3
4
   #define SIZE 40
5
6
   // Queue structure
7
   struct queue {
8
        int items[SIZE];
9
        int front;
10
        int rear;
11
   };
12
13
   // Node structure for adjacency list
14
   struct node {
15
        int vertex;
        struct node* next;
16
17
   };
18
19
   // Graph structure
20
   struct Graph {
21
        int numVertices;
22
        struct node** adjLists;
23
        int* visited;
24
   };
25
26
   // Function to create a node
27
   struct node* createNode(int v) {
        struct node* newNode = (struct node*)malloc(sizeof(struct node));
28
29
        if (!newNode) {
30
            printf("Memory allocation failed\n");
31
            exit(1);
32
33
        newNode->vertex = v;
34
        newNode->next = NULL;
35
        return newNode;
36
   }
37
   // Function to create a graph
38
39
    struct Graph* createGraph(int vertices) {
40
        struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
41
        if (!graph) {
42
            printf("Memory allocation failed\n");
            exit(1);
43
44
        }
45
46
        graph->numVertices = vertices;
        graph->adjLists = (struct node**)malloc(vertices * sizeof(struct node*));
47
48
        graph->visited = (int*)malloc(vertices * sizeof(int));
```

```
49
50
        for (int i = 0; i < vertices; i++) {</pre>
51
            graph->adjLists[i] = NULL;
52
            graph->visited[i] = 0;
53
54
        return graph;
55
    }
56
57
    // Function to add an edge to the graph
58
    void addEdge(struct Graph* graph, int src, int dest) {
59
        // Add edge from src to dest
        struct node* newNode = createNode(dest);
60
61
        newNode->next = graph->adjLists[src];
62
        graph->adjLists[src] = newNode;
63
64
        // Add edge from dest to src (since the graph is undirected)
65
        newNode = createNode(src);
        newNode->next = graph->adjLists[dest];
66
        graph->adjLists[dest] = newNode;
67
68
69
70
    // Function to create a queue
71
    struct queue* createQueue() {
        struct queue* q = (struct queue*)malloc(sizeof(struct queue));
72
73
        if (!q) {
74
            printf("Memory allocation failed\n");
75
            exit(1);
76
77
        q \rightarrow front = -1;
78
        q \rightarrow rear = -1;
79
        return q;
80
    }
81
82
    // Function to check if the queue is empty
83
    int isEmpty(struct queue* q) {
84
        return q->rear == -1;
85
    }
86
87
    // Function to add an element to the queue
    void enqueue(struct queue* q, int value) {
88
89
        if (q->rear == SIZE - 1) {
90
            printf("\nQueue is full!\n");
91
        } else {
92
            if (q->front == -1)
93
                 q \rightarrow front = 0;
94
            q->rear++;
95
            q->items[q->rear] = value;
96
        }
97
    }
98
```

```
99
    // Function to remove an element from the queue
100
     int dequeue(struct queue* q) {
101
         int item = -1;
102
         if (isEmpty(q)) {
103
             printf("Queue is empty\n");
104
         } else {
105
             item = q->items[q->front];
106
             q->front++;
107
             if (q->front > q->rear) {
108
                 q->front = q->rear = -1; // Reset queue
109
             }
110
111
         return item;
112
     }
113
114
     // Function to print the queue
115
     void printQueue(struct queue* q) {
116
         if (isEmpty(q)) {
117
             printf("Queue is empty\n");
118
         } else {
119
             printf("\nQueue contains: ");
120
             for (int i = q->front; i <= q->rear; i++) {
                 printf("%d ", q->items[i]);
121
122
             }
             printf("\n");
123
124
         }
125
     }
126
127
     // BFS algorithm
128
     void bfs(struct Graph* graph, int startVertex) {
129
         struct queue* q = createQueue();
130
131
         graph->visited[startVertex] = 1;
132
         enqueue(q, startVertex);
133
134
         while (!isEmpty(q)) {
135
             printQueue(q);
136
             int currentVertex = dequeue(q);
137
             printf("Visited %d\n", currentVertex);
138
139
             struct node* temp = graph->adjLists[currentVertex];
140
141
             while (temp) {
142
                 int adjVertex = temp->vertex;
143
144
                 if (graph->visited[adjVertex] == 0) {
145
                     graph->visited[adjVertex] = 1;
146
                     enqueue(q, adjVertex);
147
148
                 temp = temp->next;
```

```
149
             }
150
151
152
         // Free the queue
153
         free(q);
154
     }
155
    // Main function
156
157
     int main() {
158
         struct Graph* graph = createGraph(6);
159
160
         addEdge(graph, 0, 1);
161
         addEdge(graph, 0, 2);
162
         addEdge(graph, 1, 2);
163
         addEdge(graph, 1, 4);
         addEdge(graph, 1, 3);
164
165
         addEdge(graph, 2, 4);
166
         addEdge(graph, 3, 4);
167
168
         printf("BFS Traversal starting from vertex 0:\n");
169
         bfs(graph, ₀);
170
         // Free allocated memory for the graph
171
         for (int i = 0; i < graph->numVertices; i++) {
172
173
             struct node* temp = graph->adjLists[i];
174
             while (temp) {
175
                 struct node* toFree = temp;
176
                 temp = temp->next;
                 free(toFree);
177
178
             }
179
180
         free(graph->adjLists);
181
         free(graph->visited);
182
         free(graph);
183
184
         return 0;
185
    }
    /*
186
187
    Output:
188
    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 }
189
    BFS Traversal starting from vertex 0:
190
191
    Queue contains: 0
    Visited 0
192
193
194
    Queue contains: 2 1
    Visited 2
195
196
```

10/18/24, 7:02 AM

10/18/24, 7:02 AM bfs.c

197 Queue contains: 1 4
198 Visited 1
199
200 Queue contains: 4 3
201 Visited 4
202
203 Queue contains: 3
204 Visited 3

205 */