

## Q1. Adjacency matrix representation and DFS

CODE:

```
1  /*
2   |   Name: Om Vivek Gharge
3   |   PRN: 2020BTEIT00041
4   */
5
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  struct Node
10 {
11     int data;
12     struct Node *next;
13 } *front = NULL, *rear = NULL;
14 void enqueue(int x)
15 {
16     struct Node *t;
17     t = (struct Node *)malloc(sizeof(struct Node));
18     if (t == NULL)
19         printf("Queue is Full\n");
20     else
21     {
22         t->data = x;
23         t->next = NULL;
24         if (front == NULL)
25             front = rear = t;
26         else
27         {
28             rear->next = t;
29             rear = t;
30         }
31     }
32 }
33 int dequeue()
34 {
35     int x = -1;
36     struct Node *t;
37     if (front == NULL)
38         printf("Queue is Empty\n");
39     else
40     {
```

```

41         x = front->data;
42         t = front;
43         front = front->next;
44         free(t);
45     }
46     return x;
47 }
48 int isEmpty()
49 {
50     return front == NULL;
51 }
52
53 void DFS(int G[][7], int start, int n)
54 {
55     static int visited[7] = {0};
56     int j;
57     if (visited[start] == 0)
58     {
59         printf("%d ", start);
60         visited[start] = 1;
61         for (j = 1; j < n; j++)
62         {
63             if (G[start][j] == 1 && visited[j] == 0)
64                 DFS(G, j, n);
65         }
66     }
67 }
68 int main()
69 {
70     int G[7][7] = {{0, 0, 0, 0, 0, 0, 0},
71                    {0, 0, 1, 1, 0, 0, 0},
72                    {0, 1, 0, 0, 1, 0, 0},
73                    {0, 1, 0, 0, 1, 0, 0},
74                    {0, 0, 1, 1, 0, 1, 1},
75                    {0, 0, 0, 0, 1, 0, 0},
76                    {0, 0, 0, 0, 1, 0, 0}};
77
78     printf("Transversal: ");
79     DFS(G, 4, 7);
80
81     return 0;
82 }

```

OUTPUT:

Transversal: 4 2 1 3 5 6

## Q2. Adjacency matrix representation and DFS

CODE:

```
1  /*
2     Name: Om Vivek Gharge
3     PRN: 2020BTEIT00041
4  */
5
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  struct Node
10 {
11     int data;
12     struct Node *next;
13 } *front = NULL, *rear = NULL;
14 void enqueue(int x)
15 {
16     struct Node *t;
17     t = (struct Node *)malloc(sizeof(struct Node));
18     if (t == NULL)
19         printf("Queue is Full\n");
20     else
21     {
22         t->data = x;
23         t->next = NULL;
24         if (front == NULL)
25             front = rear = t;
26         else
27         {
28             rear->next = t;
29             rear = t;
30         }
31     }
32 }
33 int dequeue()
34 {
35     int x = -1;
36     struct Node *t;
37     if (front == NULL)
38         printf("Queue is Empty\n");
39     else
40     {
```

```

41     x = front->data;
42     t = front;
43     front = front->next;
44     free(t);
45 }
46 return x;
47 }
48 int isEmpty()
49 {
50     return front == NULL;
51 }
52
53 void BFS(int G[][7], int start, int n)
54 {
55     int i = start, j;
56     int visited[7] = {0};
57     printf("%d ", i);
58     visited[i] = 1;
59     enqueue(i);
60     while (!isEmpty())
61     {
62         i = dequeue();
63         for (j = 1; j < n; j++)
64         {
65             if (G[i][j] == 1 && visited[j] == 0)
66             {
67                 printf("%d ", j);
68                 visited[j] = 1;
69                 enqueue(j);
70             }
71         }
72     }
73 }
74
75 int main()
76 {
77     int G[7][7] = {{0, 0, 0, 0, 0, 0, 0},
78                   {0, 0, 1, 1, 0, 0, 0},
79                   {0, 1, 0, 0, 1, 0, 0},
80                   {0, 1, 0, 0, 1, 0, 0},
81                   {0, 0, 1, 1, 0, 1, 1},
82                   {0, 0, 0, 0, 1, 0, 0},
83                   {0, 0, 0, 0, 1, 0, 0}};
84
85     printf("Transversal: ");
86     BFS(G, 4, 7);
87     return 0;
88 }

```

CODE:

```

Transversal: 4 2 3 5 6 1

```

### Q3. Adjacency list representation and DFS

CODE:

```
1  /*
2     Name: Om Vivek Gharge
3     PRN: 2020BTEIT00041
4  */
5
6  #include <stdio.h>
7
8  #include <stdlib.h>
9
10 #include <string.h>
11
12 typedef struct QNode
13 {
14     void *data;
15     struct QNode *next;
16 } QNode;
17
18 typedef struct
19 {
20     QNode *front;
21     QNode *rear;
22 } Queue;
23
24 void enqueue(Queue *q, void *data)
25 {
26     QNode *temp = (QNode *)malloc(sizeof(QNode));
27     if (!temp)
28     {
29         printf("Queue is Full\n");
30     }
31 }
```

```

41         return;
42     }
43
44     temp->data = data;
45
46     temp->next = NULL;
47
48     if (!q->front)
49     {
50
51         // Empty Q
52
53         q->front = q->rear = temp;
54     }
55     else
56     {
57
58         q->rear->next = temp;
59
60         q->rear = temp;
61     }
62 }
63
64 void *dequeue(Queue *q)
65 {
66
67     void *data = NULL;
68
69     QNode *temp = NULL;
70
71     if (q->front)
72     {
73
74         temp = q->front;
75
76         q->front = q->front->next;
77
78         data = temp->data;
79
80

```

```

80
81     free(temp);
82
83     temp = NULL;
84 }
85 else
86 {
87
88     printf("Queue is Empty\n");
89 }
90
91 return data;
92 }
93
94 int isEmpty(Queue *q)
95 {
96
97
98     if (!q->front)
99         return 1;
100
101     else
102         return 0;
103 }
104
105 /* Graph Data structure */
106
107 typedef struct Vertex
108 {
109
110     int vertex;
111
112     int weight;
113
114     struct Vertex *next;
115 } Vertex;
116
117
118 typedef struct
119 {

```

```

120
121     int no_of_nodes;
122
123     Vertex **List;
124
125 } Graph;
126
127 int search_graph[][6] = {
128     {0, 1, 1, 0, 0, 0},
129     {1, 0, 0, 1, 0, 0},
130     {1, 0, 0, 1, 0, 0},
131     {0, 1, 1, 0, 1, 1},
132     {0, 0, 0, 1, 0, 0},
133     {0, 0, 0, 1, 0, 0}
134 };
135
136 Graph *createGraph(int nodes)
137 {
138     int i = 0;
139     int edges;
140
141     Vertex *v = NULL;
142
143     Vertex *lastv = NULL;
144
145     Graph *G = (Graph *)malloc(sizeof(Graph));
146
147     if (G && nodes > 0)
148
149
150
151
152
153
154
155
156
157
158

```



```

{

    G->no_of_nodes = 0;

    G->List = (Vertex **)malloc(sizeof(Vertex *) * nodes);

    if (G->List)
    {
        for (i = 0; i < nodes; i++)
        {
            G->List[i] = NULL;

            ++G->no_of_nodes;

            for (edges = 0; edges < nodes; edges++)
            {
                if (!search_graph[i][edges])
                    continue;

                v = (Vertex *)malloc(sizeof(Vertex));

                if (!v)
                    return NULL;

                v->vertex = edges;
                v->weight = 0;
                v->next = NULL;

                if (!G->List[i])
                    G->List[i] = v;

                else
                    lastv->next = v;

                lastv = v;
            }
        }
    }
}

```

```

199     }
200 }
201
202     return G;
203 }
204
205 void displayGraph(Graph *G)
206 {
207     int node;
208
209     Vertex *v = NULL;
210
211     if (!G || G->no_of_nodes <= 0)
212         return;
213
214     for (node = 0; node < G->no_of_nodes; ++node)
215     {
216         printf("\n Node(%d) \n", node);
217
218         if (G->List[node])
219         {
220             v = G->List[node];
221
222             while (v)
223             {
224                 printf("Vertex = %d (%d)", v->vertex, v->weight);
225
226                 v = v->next;
227
228                 if (v)
229                     printf(" -> ");
230             }
231
232             v = NULL;
233         }
234     }
235 }
236
237

```

```

238     }
239 }
240 }
241
242 void DFS(Graph *G, int node)
243 {
244     Vertex *v = NULL;
245
246     //static int visited[6] = {0};
247
248     static int *visited = NULL;
249
250     if (!visited)
251     {
252         visited = (int *)malloc(sizeof(int) * G->no_of_nodes);
253         memset(visited, 0, sizeof(int) * G->no_of_nodes);
254     }
255
256     if (!G || !G->List)
257         return;
258
259     printf(" %d ", node);
260
261     ++visited[node];
262
263     v = G->List[node];
264
265     while (v)
266     {
267         if (!visited[v->vertex])
268             DFS(G, v->vertex);
269
270         v = v->next;
271     }
272 }

```

```

278     }
279
280
281
282     int main()
283     {
284
285         Graph *G = NULL;
286
287         int nodes = 6;
288
289         if (nodes > 0)
290         {
291
292             G = createGraph(nodes);
293
294             // displayGraph(G);
295
296             printf("\nTransversal: ");
297             DFS(G, 3);
298
299
300         }
301
302         return 0;
303     }
304

```

OUTPUT:

```

Transversal:  3  1  0  2  4  5

```

#### Q4. Adjacency list representation and BFS

CODE:

```
1  /*
2  |   Name: Om Vivek Gharge
3  |   PRN: 2020BTEIT00041
4  */
5
6  #include <stdio.h>
7
8  #include <stdlib.h>
9
10 #include <string.h>
11
12 typedef struct QNode
13 {
14     void *data;
15
16     struct QNode *next;
17 } QNode;
18
19 typedef struct
20 {
21     QNode *front;
22
23     QNode *rear;
24 } Queue;
25
26 void enqueue(Queue *q, void *data)
27 {
28     QNode *temp = (QNode *)malloc(sizeof(QNode));
29
30     if (!temp)
31     {
32         printf("Queue is Full\n");
33     }
34 }
```

```
41     return;
42 }
43
44 temp->data = data;
45
46 temp->next = NULL;
47
48 if (!q->front)
49 {
50     // Empty Q
51
52     q->front = q->rear = temp;
53 }
54 else
55 {
56     q->rear->next = temp;
57
58     q->rear = temp;
59 }
60 }
61
62 }
63
64 void *dequeue(Queue *q)
65 {
66     void *data = NULL;
67
68     QNode *temp = NULL;
69
70     if (q->front)
71     {
72         temp = q->front;
73
74         q->front = q->front->next;
75
76         data = temp->data;
77     }
78
79 }
```

```

80
81     free(temp);
82
83     temp = NULL;
84 }
85 else
86 {
87
88     printf("Queue is Empty\n");
89 }
90
91 return data;
92 }
93
94 int isEmpty(Queue *q)
95 {
96
97
98     if (!q->front)
99         return 1;
100
101     else
102         return 0;
103 }
104
105 /* Graph Data structure */
106
107 typedef struct Vertex
108 {
109
110     int vertex;
111
112     int weight;
113
114     struct Vertex *next;
115
116 } Vertex;
117
118 typedef struct
119 {
120

```

```
120
121     int no_of_nodes;
122
123     Vertex **List;
124 } Graph;
125
126 int search_graph[][6] = {
127     {0, 1, 1, 0, 0, 0},
128     {1, 0, 0, 1, 0, 0},
129     {1, 0, 0, 1, 0, 0},
130     {0, 1, 1, 0, 1, 1},
131     {0, 0, 0, 1, 0, 0},
132     {0, 0, 0, 1, 0, 0}
133 };
134
135 Graph *createGraph(int nodes)
136 {
137     int i = 0;
138     int edges;
139     Vertex *v = NULL;
140     Vertex *lastv = NULL;
141     Graph *G = (Graph *)malloc(sizeof(Graph));
142     if (G && nodes > 0)
143     {
```



```

160
161     G->no_of_nodes = 0;
162
163     G->List = (Vertex **)malloc(sizeof(Vertex *) * nodes);
164
165     if (G->List)
166     {
167
168         for (i = 0; i < nodes; i++)
169         {
170
171             G->List[i] = NULL;
172
173             ++G->no_of_nodes;
174
175             for (edges = 0; edges < nodes; edges++)
176             {
177
178                 if (!search_graph[i][edges])
179                     continue;
180
181                 v = (Vertex *)malloc(sizeof(Vertex));
182
183                 if (!v)
184                     return NULL;
185
186                 v->vertex = edges;
187                 v->weight = 0;
188                 v->next = NULL;
189
190                 if (!G->List[i])
191                     G->List[i] = v;
192
193                 else
194                     lastv->next = v;
195
196                 lastv = v;
197             }
198         }
199     }

```

```

200     }
201
202     return G;
203 }
204
205 void displayGraph(Graph *G)
206 {
207     int node;
208
209     Vertex *v = NULL;
210
211     if (!G || G->no_of_nodes <= 0)
212         return;
213
214     for (node = 0; node < G->no_of_nodes; ++node)
215     {
216         printf("\n Node(%d) \n", node);
217
218         if (G->List[node])
219         {
220             v = G->List[node];
221
222             while (v)
223             {
224                 printf("Vertex = %d (%d)", v->vertex, v->weight);
225
226                 v = v->next;
227
228                 if (v)
229                     printf(" -> ");
230             }
231
232             v = NULL;
233         }
234     }
235 }
236
237 }
238
239 }

```

```

240 }
241
242 void BFS(Graph *G, int node)
243 {
244     Queue q = {NULL, NULL};
245
246     Vertex *v;
247
248     static int *visited = NULL;
249
250     int *next;
251
252     if (!G || !G->List)
253         return;
254
255     if (!visited)
256     {
257         visited = (int *)malloc(sizeof(int) * G->no_of_nodes);
258         memset(visited, 0, sizeof(int) * G->no_of_nodes);
259     }
260
261     printf(" %d ", node);
262
263     ++visited[node];
264
265     enqueue(&q, (void *)G->List[node]);
266
267     while (!isEmpty(&q))
268     {
269         v = (Vertex *)dequeue(&q);
270
271         while (v)
272         {
273

```

```

279         if (!visited[v->vertex])
280         {
281
282             printf(" %d ", v->vertex);
283
284             ++visited[v->vertex];
285
286             enqueue(&q, (void *)G->List[v->vertex]);
287         }
288
289         v = v->next;
290     }
291 }
292 }
293
294 int main()
295 {
296
297     Graph *G = NULL;
298
299     int nodes = 6;
300
301     if (nodes > 0)
302     {
303
304         G = createGraph(nodes);
305
306         // displayGraph(G);
307
308         printf("\nTransversal: ");
309         BFS(G, 3);
310     }
311
312     return 0;
313 }
314

```

OUTPUT:

```
Transversal: 3 1 2 4 5 0
```