

HW 3

2018312164 김석진

코드

```
C main.c U X
Data_Structure > assignment5 > C main.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <limits.h>
4
5  typedef struct _graph {
6  int vertex;
7  int length;
8  struct _graph* link;
9  } linked_adja;
10 /*
11 static int v_num = 8;
12 static int adjacency_matrix[8][8] = {
13 {0, 0, 47, 0, 70, 24, 0, 0},
14 {0, 0, 0, 31, 0, 0, 74, 79},
15 {0, 55, 0, 88, 23, 0, 66, 0},
16 {0, 0, 0, 0, 0, 0, 0, 29},
17 {0, 31, 0, 0, 0, 0, 42, 0},
18 {0, 0, 25, 120, 0, 0, 0, 0},
19 {0, 0, 0, 0, 0, 0, 0, 66},
20 {0, 0, 0, 0, 0, 0, 0, 0}
21 };
22 */
23 static int v_num = 8;
24 static int adjacency_matrix[8][8] = {
25 {0, 0, 0, 0, 0, 0, 0, 0},
26 {300, 0, 0, 0, 0, 0, 0, 0},
27 {1000, 800, 0, 0, 0, 0, 0, 0},
28 {0, 0, 1200, 0, 0, 0, 0, 0},
29 {0, 0, 0, 1500, 0, 250, 0, 0},
30 {0, 0, 0, 1000, 0, 0, 900, 1400},
31 {0, 0, 0, 0, 0, 0, 0, 1000},
32 {1700, 0, 0, 0, 0, 0, 0, 0}
33 };
34
35 void ssadsp(linked_adja** graph_array, int start_vertex, int **vertex_log, int *distance_matrix) {
36 // Initialize distance_matrix and vertex_log
37 for (int i = 0; i < v_num; i++) {
38 distance_matrix[i] = INT_MAX; // Set all distances to infinity
39 for (int j = 0; j < v_num; j++) {
40 vertex_log[i][j] = 0;
41 }
42 }
43
44 // Set distance to the start_vertex as 0
45 distance_matrix[start_vertex] = 0;
46
47 // Iterate through all vertices
48 for (int k = 0; k < v_num - 1; k++) {
49 for (int i = 0; i < v_num; i++) {
50 // Iterate through the adjacency list of the current vertex
51 linked_adja* current = graph_array[i];
52 while (current != NULL) {
53 int j = current->vertex;
54 int weight = adjacency_matrix[i][j];
55
56 // Relaxation step
57 if (distance_matrix[i] != INT_MAX && distance_matrix[i] + weight < distance_matrix[j])
58 distance_matrix[j] = distance_matrix[i] + weight;
59
60 // Update vertex log
```

```

C main.c U X
Data_Structure > assignment5 > C main.c > ssadsp(linked_adja **,int,int **,int *)
46
47 // Iterate through all vertices
48 for (int k = 0; k < v_num - 1; k++) {
49     for (int i = 0; i < v_num; i++) {
50         // Iterate through the adjacency list of the current vertex
51         linked_adja* current = graph_array[i];
52         while (current != NULL) {
53             int j = current->vertex;
54             int weight = adjacency_matrix[i][j];
55
56             // Relaxation step
57             if (distance_matrix[i] != INT_MAX && distance_matrix[i] + weight < distance_matrix[j])
58                 distance_matrix[j] = distance_matrix[i] + weight;
59
60             // Update vertex_log
61             int l;
62             for (l = 0; vertex_log[i][l] != 0; l++) {
63                 vertex_log[j][l] = vertex_log[i][l];
64             }
65             vertex_log[j][l] = i + 1; // +1 to convert to 1-based indexing
66         }
67
68         current = current->link;
69     }
70 }
71
72
73 for(int j = 0; j < v_num; j++) {
74     if(distance_matrix[j] == INT_MAX) {
75         distance_matrix[j] = 0;
76     }
77 }
78
79 }
80
81 int main(void) {
82
83     linked_adja** graph_array = (linked_adja**)malloc(v_num * sizeof(linked_adja*));
84
85     for (int i = 0; i < v_num; i++) {
86         graph_array[i] = NULL; // Initialize each element to NULL
87
88         for(int j = 0; j < v_num; j++) {
89             int temp = adjacency_matrix[i][j];
90             if(temp > 0) {
91                 linked_adja* node = (linked_adja*) malloc(sizeof(linked_adja));
92                 node->vertex = j;
93                 node->link = NULL;
94
95                 if (graph_array[i] == NULL) {
96                     // If the list is empty, make the new node the head of the list
97                     graph_array[i] = node;
98                 } else {
99                     // Find the end of the list and add the new node
100                     linked_adja* end_node = graph_array[i];
101                     while(end_node->link != NULL) {
102                         end_node = end_node->link;
103                     }
104
105                     end_node->link = node;
106                 }
107             }
108         }
109     }
110 }

```

```

C main.c U X
Data_Structure > assignment5 > C main.c > ...
74 int main(void) {
75
76     linked_adja** graph_array = (linked_adja**)malloc(v_num * sizeof(linked_adja*));
77
78     for (int i = 0; i < v_num; i++) {
79         graph_array[i] = NULL; // Initialize each element to NULL
80
81         for(int j = 0; j < v_num; j++) {
82             int temp = adjacency_matrix[i][j];
83             if(temp > 0) {
84                 linked_adja* node = (linked_adja*) malloc(sizeof(linked_adja));
85                 node->vertex = j;
86                 node->link = NULL;
87
88                 if (graph_array[i] == NULL) {
89                     // If the list is empty, make the new node the head of the list
90                     graph_array[i] = node;
91                 } else {
92                     // Find the end of the list and add the new node
93                     linked_adja* end_node = graph_array[i];
94                     while(end_node->link != NULL) {
95                         end_node = end_node->link;
96                     }
97                     end_node->link = node;
98                 }
99             }
100         }
101     }
102
103     // single source all destination shortest paths
104     int start_vertex = 4;
105     int* distance_matrix = (int*)malloc(v_num * sizeof(int));
106     int** vertex_log = (int**)malloc(v_num * sizeof(int*));
107
108     for (int v_l_i = 0; v_l_i < v_num; v_l_i++) {
109         vertex_log[v_l_i] = (int*)malloc(v_num * sizeof(int));
110     }
111
112     ssadsp(graph_array, start_vertex, vertex_log, distance_matrix);
113
114     // print
115     for (int p_i = 0; p_i < v_num; p_i++) {
116         printf("%d vertex log : ", p_i);
117
118         for (int log_i = 0; log_i < v_num; log_i++) {
119             if (vertex_log[p_i][log_i] == 0) {
120                 break;
121             }
122             printf("%d ", vertex_log[p_i][log_i]);
123         }
124
125         printf("distance : %d\n", distance_matrix[p_i]);
126     }
127
128     return 0;
129 }
130
131
132
133

```

코드 설명

```
void ssadsp(linked_adja** graph_array, int start_vertex, int **vertex_log, int *distance_matrix) {
    // Initialize distance_matrix and vertex_log
    for (int i = 0; i < v_num; i++) {
        distance_matrix[i] = INT_MAX; // Set all distances to infinity
        for (int j = 0; j < v_num; j++) {
            vertex_log[i][j] = 0;
        }
    }

    // Set distance to the start vertex as 0
    distance_matrix[start_vertex] = 0;

    // Iterate through all vertices
    for (int k = 0; k < v_num - 1; k++) {
        for (int i = 0; i < v_num; i++) {
            // Iterate through the adjacency list of the current vertex
            linked_adja* current = graph_array[i];
            while (current != NULL) {
                int j = current->vertex;
                int weight = adjacency_matrix[i][j];

                // Relaxation step
                if (distance_matrix[i] != INT_MAX && distance_matrix[i] + weight < distance_matrix[j])
                    distance_matrix[j] = distance_matrix[i] + weight;

                // Update vertex_log
                int l;
                for (l = 0; vertex_log[i][l] != 0; l++) {
                    vertex_log[j][l] = vertex_log[i][l];
                }
                vertex_log[j][l] = i + 1; // +1 to convert to 1-based indexing

                current = current->link;
            }
        }
    }

    for(int j = 0; j < v_num; j++) {
        if(distance_matrix[j] == INT_MAX) {
            distance_matrix[j] = 0;
        }
    }
}
```

처음에 distance matrix와 vertex log을 모두 초기화 한다. 최소값을 찾고자 하기 때문에 distance_matrix의 모든 값은 최대값으로 저장되어야 한다.

```
// Initialize distance_matrix and vertex_log
for (int i = 0; i < v_num; i++) {
    distance_matrix[i] = INT_MAX; // Set all distances to infinity
    for (int j = 0; j < v_num; j++) {
        vertex_log[i][j] = 0;
    }
}
```

시작하는 노드에서 시작하는 노드로 가는 경로의 거리는 0으로 만든다.

```

// Iterate through all vertices
for (int k = 0; k < v_num - 1; k++) {
    for (int i = 0; i < v_num; i++) {
        // Iterate through the adjacency list of the current vertex
        linked_adja* current = graph_array[i];
        while (current != NULL) {
            int j = current->vertex;
            int weight = adjacency_matrix[i][j];

            // Relaxation step
            if (distance_matrix[i] != INT_MAX && distance_matrix[i] + weight < distance_matrix[j])
                distance_matrix[j] = distance_matrix[i] + weight;

            // Update vertex_log
            int l;
            for (l = 0; vertex_log[i][l] != 0; l++) {
                vertex_log[j][l] = vertex_log[i][l];
            }
            vertex_log[j][l] = i + 1; // +1 to convert to 1-based indexing
        }

        current = current->link;
    }
}
}

```

모든 노드를 for문을 이용해 돌면서 **distance**와 **vertex log**을 최신헌다. 가장 최소의 경로가 되게 업데이트를 한다.

```

for(int j = 0; j < v_num; j++) {
    if(distance_matrix[j] == INT_MAX) {
        distance_matrix[j] = 0;
    }
}

```

경로가 없는 **distance**는 0으로 변경해준다.

```

74 int main(void) {
75
76     linked_adja** graph_array = (linked_adja**)malloc(v_num * sizeof(linked_adja*));
77
78     for (int i = 0; i < v_num; i++) {
79         graph_array[i] = NULL; // Initialize each element to NULL
80
81         for(int j = 0; j < v_num; j++) {
82             int temp = adjacency_matrix[i][j];
83             if(temp > 0) {
84                 linked_adja* node = (linked_adja*) malloc(sizeof(linked_adja));
85                 node->vertex = j;
86                 node->link = NULL;
87
88                 if (graph_array[i] == NULL) {
89                     // If the list is empty, make the new node the head of the list
90                     graph_array[i] = node;
91                 } else {
92                     // Find the end of the list and add the new node
93                     linked_adja* end_node = graph_array[i];
94                     while(end_node->link != NULL) {
95                         end_node = end_node->link;
96                     }
97
98                     end_node->link = node;
99                 }
100             }
101         }
102     }

```

matrix에 저장되어 있는 element를 list로 옮기는 코드이다. graph array를 만들어서 list의 pointer를 저장하게 한다.

```

linked_adja** graph_array = (linked_adja**)malloc(v_num * sizeof(linked_adja*));

```

node를 생성하고 그 노드에 vertex 정보와 link 정보를 담는다. 초기에는 link의 마지막에 insert를 진행할 것이기 때문에 null을 저장한다.

```

for (int i = 0; i < v_num; i++) {
    graph_array[i] = NULL; // Initialize each element to NULL

    for(int j = 0; j < v_num; j++) {
        int temp = adjacency_matrix[i][j];
        if(temp > 0) {
            linked_adja* node = (linked_adja*) malloc(sizeof(linked_adja));
            node->vertex = j;
            node->link = NULL;

```

list가 empty인지 아닌지를 확인해서 element를 차례로 넣어준다.

```

if (graph_array[i] == NULL) {
    // If the list is empty, make the new node the head of the list
    graph_array[i] = node;
} else {
    // Find the end of the list and add the new node
    linked_adja* end_node = graph_array[i];
    while(end_node->link != NULL) {
        end_node = end_node->link;
    }

    end_node->link = node;
}

```

result

1)

```
static int adjacency_matrix[8][8] = {
{0, 0, 0, 0, 0, 0, 0, 0},
{300, 0, 0, 0, 0, 0, 0, 0},
{1000, 800, 0, 0, 0, 0, 0, 0},
{0, 0, 1200, 0, 0, 0, 0, 0},
{0, 0, 0, 1500, 0, 250, 0, 0},
{0, 0, 0, 1000, 0, 0, 900, 1400},
{0, 0, 0, 0, 0, 0, 0, 1000},
{1700, 0, 0, 0, 0, 0, 0, 0}
};
```

```
7 vertex log : 5 6      distance : 1050
daisy@daisy-15Z980-HA7WK:~/dev/TIL/Data_Structure/assignment5$ gcc main.c
daisy@daisy-15Z980-HA7WK:~/dev/TIL/Data_Structure/assignment5$ ./a.out
0 vertex log : 5 6 8      distance : 3350
1 vertex log : 5 6 4 3      distance : 3250
2 vertex log : 5 6 4      distance : 2450
3 vertex log : 5 6      distance : 1250
4 vertex log :      distance : 0
5 vertex log : 5      distance : 250
6 vertex log : 5 6      distance : 1150
7 vertex log : 5 6      distance : 1650
daisy@daisy-15Z980-HA7WK:~/dev/TIL/Data_Structure/assignment5$
```

2)

start vertex 0

```
10
11 static int v_num = 8;
12 static int adjacency_matrix[8][8] = {
13 {0, 0, 47, 0, 70, 24, 0, 0},
14 {0, 0, 0, 31, 0, 0, 74, 79},
15 {0, 55, 0, 88, 23, 0, 66, 0},
16 {0, 0, 0, 0, 0, 0, 0, 29},
17 {0, 31, 0, 0, 0, 0, 42, 0},
18 {0, 0, 25, 120, 0, 0, 0, 0},
19 {0, 0, 0, 0, 0, 0, 0, 66},
20 {0, 0, 0, 0, 0, 0, 0, 0}
21 };
22
```

```
23 //static int v_num = 8;
24 // static int adjacency_matrix[8][8] = {
25 // {0, 0, 0, 0, 0, 0, 0, 0},
26 // {300, 0, 0, 0, 0, 0, 0, 0},
27 // {1000, 800, 0, 0, 0, 0, 0, 0},
28 // {0, 0, 1200, 0, 0, 0, 0, 0},
29 // {0, 0, 0, 1500, 0, 250, 0, 0},
30 // {0, 0, 0, 1000, 0, 0, 900, 1400},
31 // {0, 0, 0, 0, 0, 0, 0, 1000},
32 // {1700, 0, 0, 0, 0, 0, 0, 0}
33 // };
34
```

```
daisy@daisy-15Z980-HA7WK:~/dev/TIL/Data_Structure/assignment5$ gcc main.c
daisy@daisy-15Z980-HA7WK:~/dev/TIL/Data_Structure/assignment5$ ./a.out
0 vertex log : distance : 0
1 vertex log : 1 5 distance : 101
2 vertex log : 1 distance : 47
3 vertex log : 1 5 2 distance : 132
4 vertex log : 1 distance : 70
5 vertex log : 1 distance : 24
6 vertex log : 1 5 distance : 112
7 vertex log : 1 5 2 4 distance : 161
daisy@daisy-15Z980-HA7WK:~/dev/TIL/Data_Structure/assignment5$
```