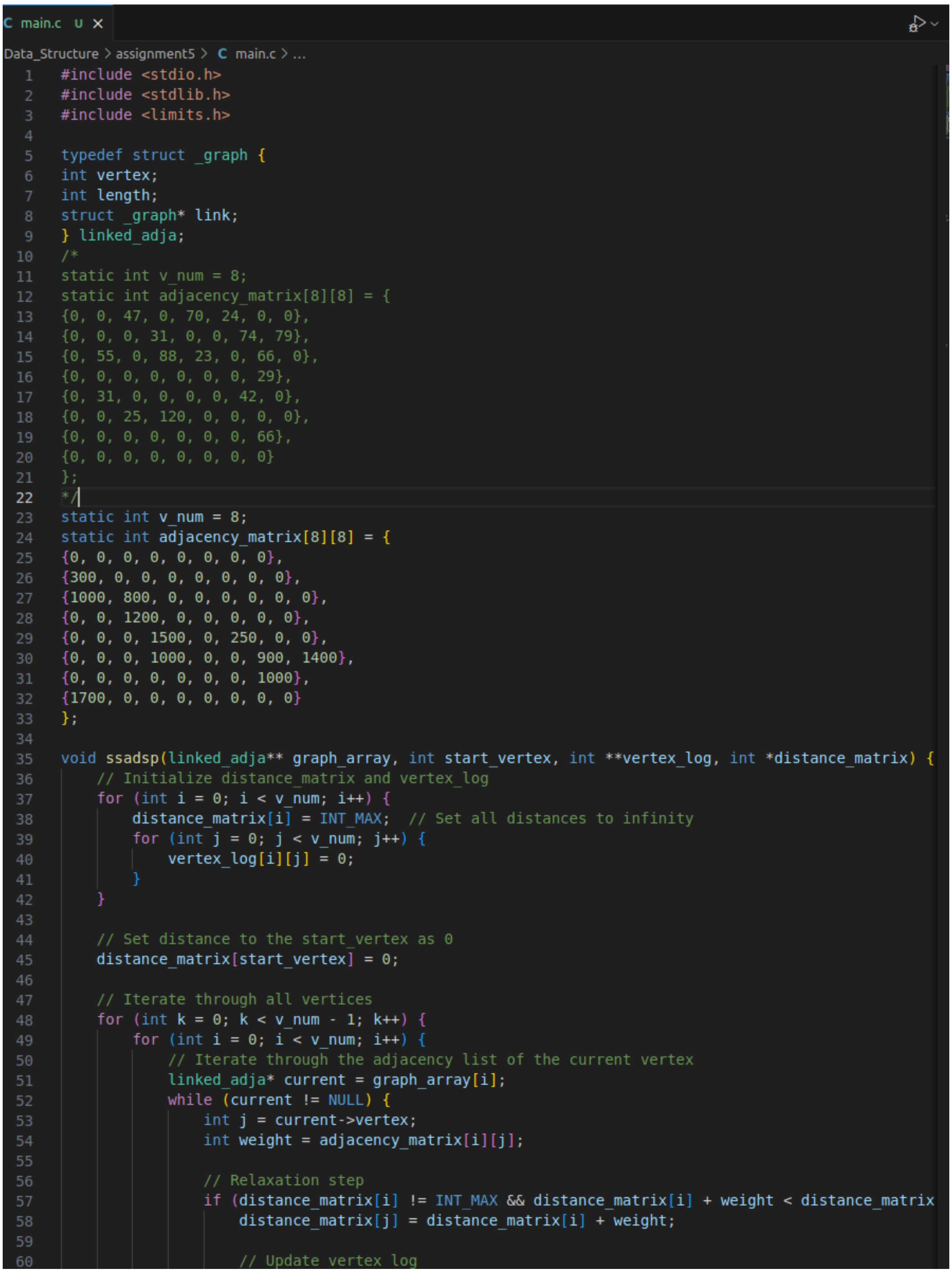
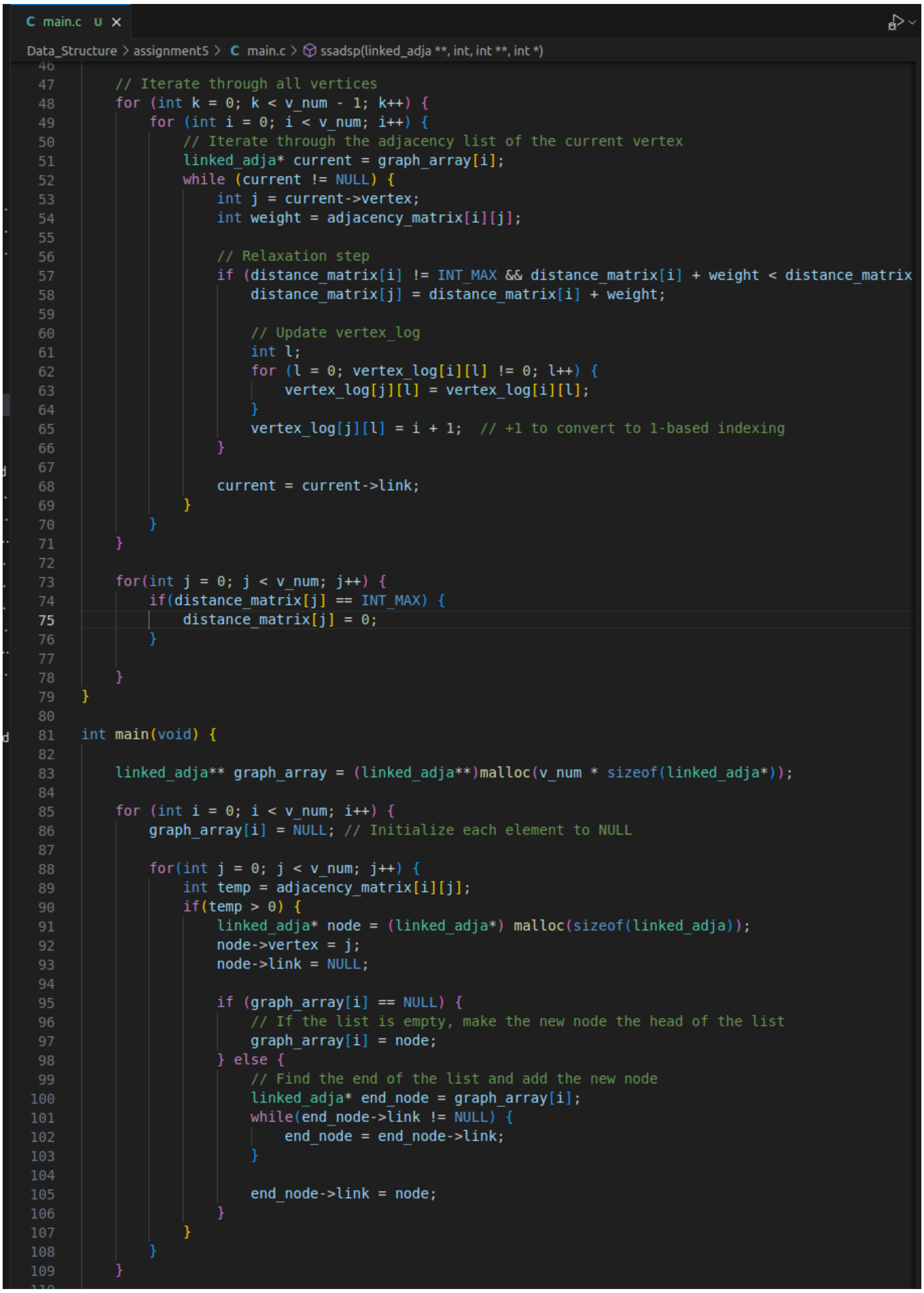
HW 3

2018312164 김석진

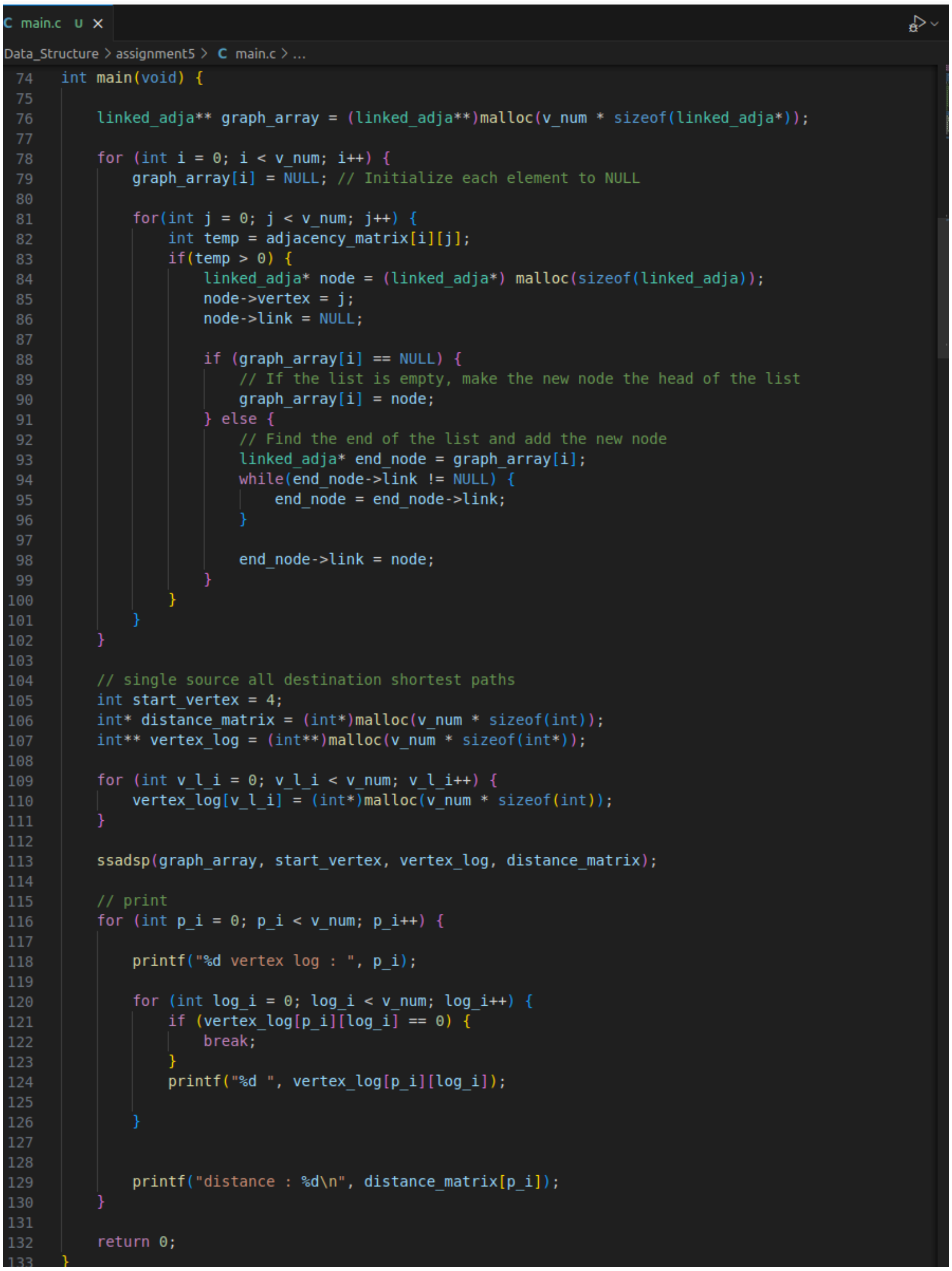
코드



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

typedef struct _graph {
int vertex;
int length;
struct _graph* link;
} linked_adja;
/*
static int v_num = 8;
static int adjacency_matrix[8][8] = {
{0, 0, 47, 0, 70, 24, 0, 0},
{0, 0, 0, 31, 0, 0, 74, 79},
{0, 55, 0, 88, 23, 0, 66, 0},
{0, 0, 0, 0, 0, 0, 0, 29},
{0, 31, 0, 0, 0, 0, 42, 0},
{0, 0, 25, 120, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 66},
{0, 0, 0, 0, 0, 0, 0, 0}
};
*/
static int v_num = 8;
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}
};

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
                    distance_matrix[j] = distance_matrix[i] + weight;

                    // Update vertex log
```

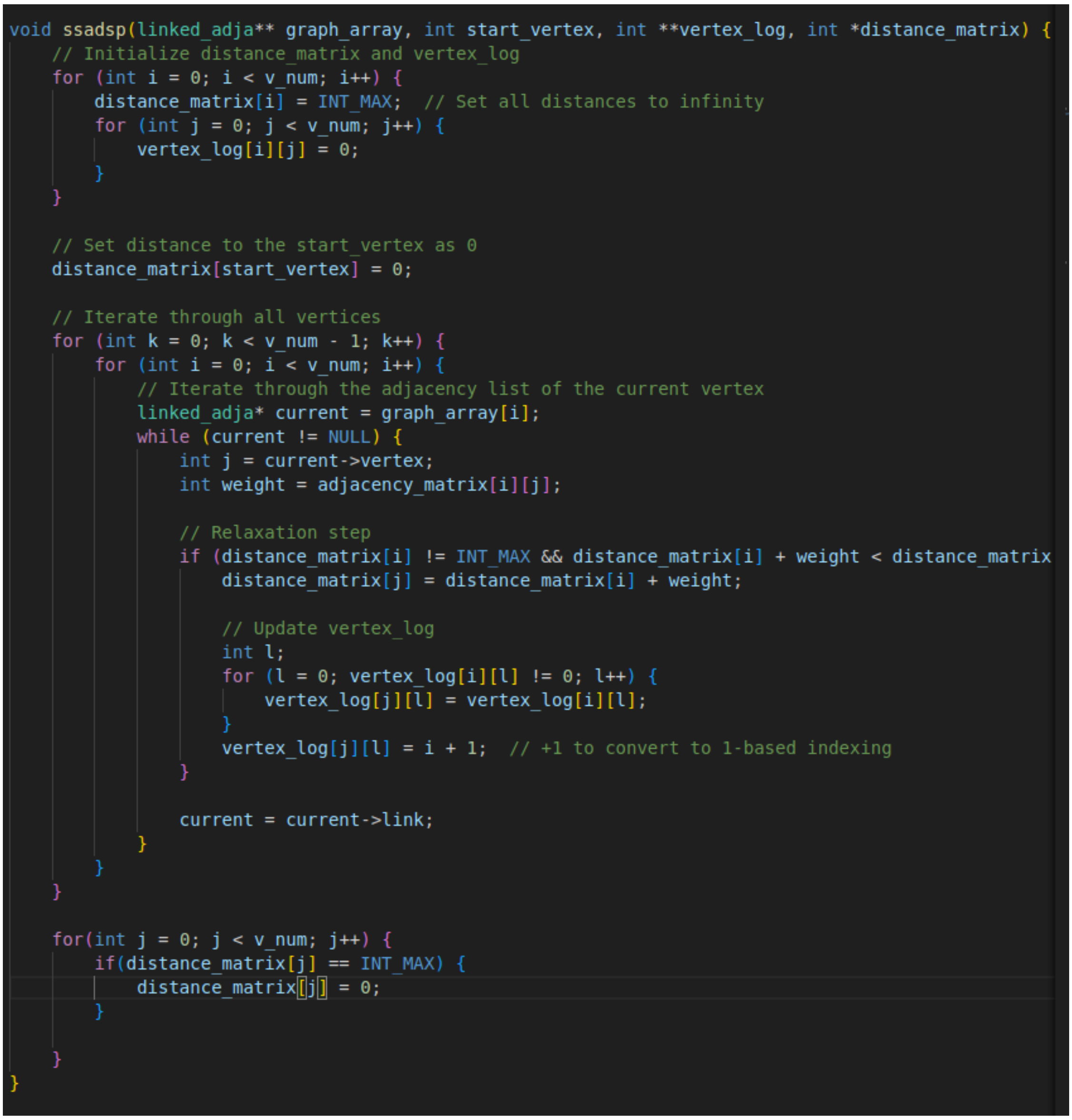

C main.c U ×
Data_Structure > assignment5 > C main.c > ssadsp(linked_adja **, int, int **, int *)

```c
    // 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
                    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;
        }

    }
}

int main(void) {

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

    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;

                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;
                }
            }
        }
    }
```



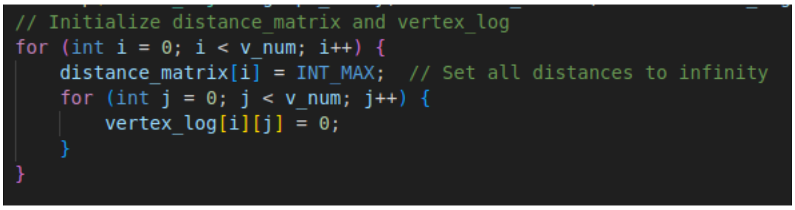
```c
int main(void) {

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

    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;

                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;
                }
            }
        }
    }

    // single source all destination shortest paths
    int start_vertex = 4;
    int* distance_matrix = (int*)malloc(v_num * sizeof(int));
    int** vertex_log = (int**)malloc(v_num * sizeof(int*));

    for (int v_l_i = 0; v_l_i < v_num; v_l_i++) {
        vertex_log[v_l_i] = (int*)malloc(v_num * sizeof(int));
    }

    ssadsp(graph_array, start_vertex, vertex_log, distance_matrix);

    // print
    for (int p_i = 0; p_i < v_num; p_i++) {

        printf("%d vertex log : ", p_i);

        for (int log_i = 0; log_i < v_num; log_i++) {
            if (vertex_log[p_i][log_i] == 0) {
                break;
            }
            printf("%d ", vertex_log[p_i][log_i]);

        }


        printf("distance : %d\n", distance_matrix[p_i]);
    }

    return 0;
}
```

코드 설명



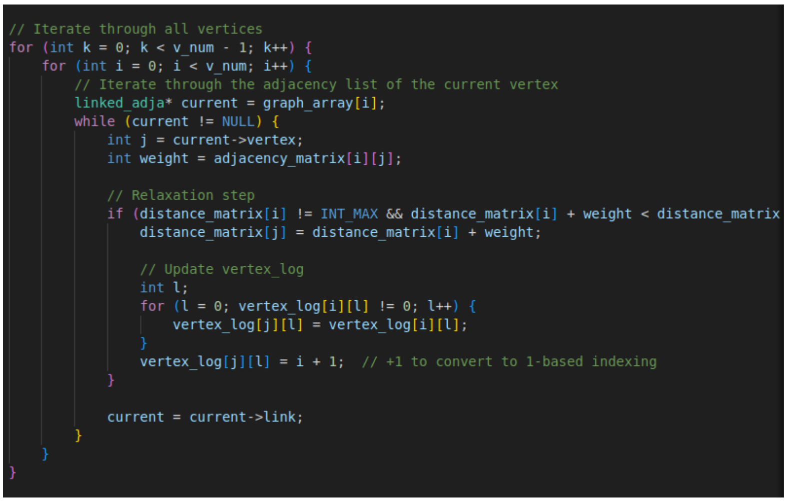
```
void ssadsp(linked_adja** graph_array, int start_vertex, int **vertex_log, int *distance_matrix) {
    // Initialize distance_matrix and vertex_log
    for (int i = 0; i < v_num; i++) {
        distance_matrix[i] = INT_MAX;  // Set all distances to infinity
        for (int j = 0; j < v_num; j++)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



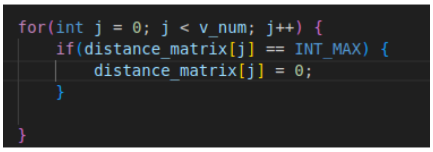
```
// Initialize distance_matrix and vertex_log
for (int i = 0; i < v_num; i++) {
    distance_matrix[i] = INT_MAX;  // Set all distances to infinity
    for (int j = 0; j < v_num; j++) {
        vertex_log[i][j] = 0;
    }
}
```

시작하는 노드에서 시작하는 노드로 가는 경로의 거리는 0으로 만든다.



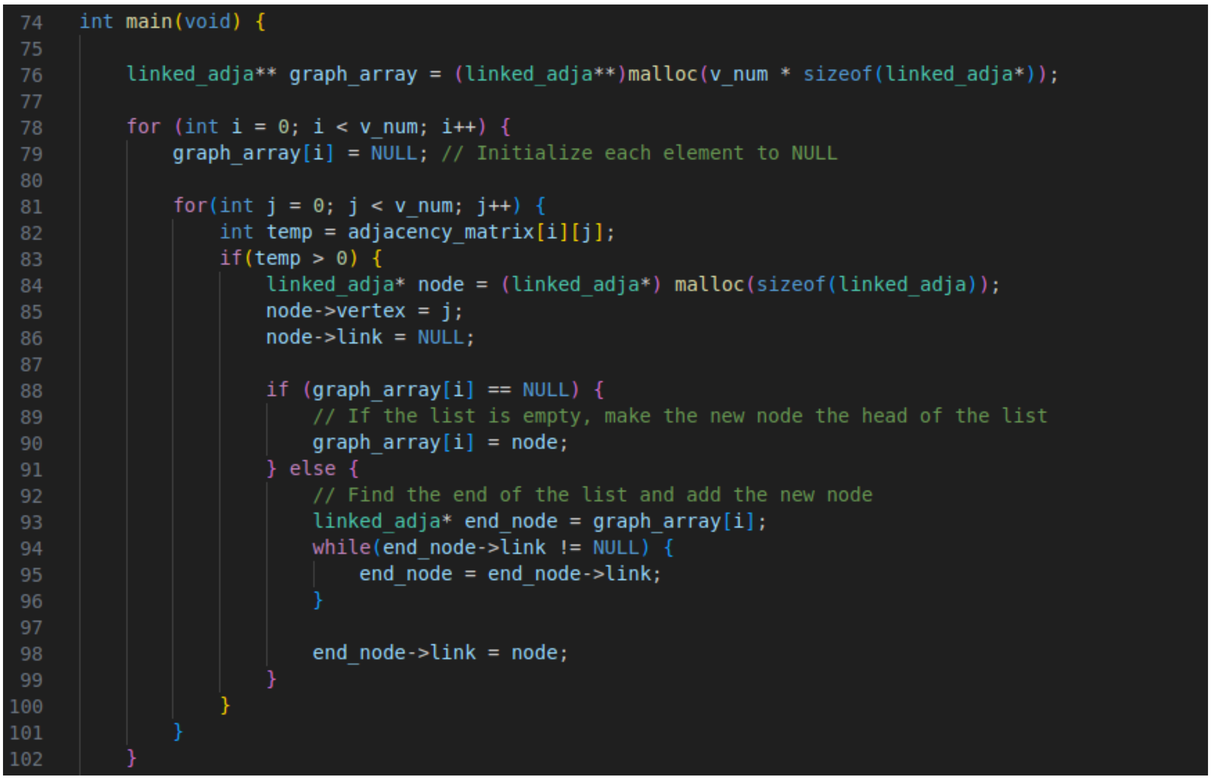
```
// Iterate through all vertices
for (int k = 0; k < v_num - 1; k++) {
    for (int i = 0; i < v_num; i++) {
        // Iterate through the adjacency list of the current vertex
        linked_adja* current = graph_array[i];
        while (current != NULL) {
            int j = current->vertex;
  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

            // Relaxation step
            if (distance_matrix[i] != INT_MAX && distance_matrix[i] + weight < distance_matrix
                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 distance와 vertex log을 최신화 한다. 가장 최소의 경로가 되게 업데이트를 한다.



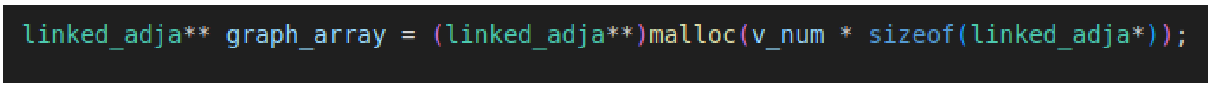
```
for(int j = 0; j < v_num; j++) {
    if(distance_matrix[j] == INT_MAX) {
        distance_matrix[j] = 0;
    }

}
```

경로가 없는 distance는 0으로 변경해준다.



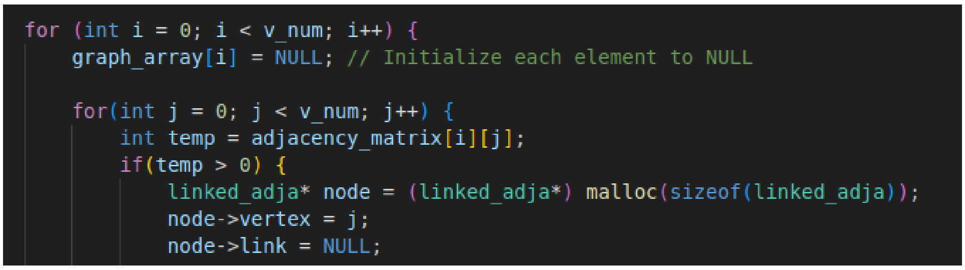
```c
int main(void) {

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

    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;

                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;
                }
            }
        }
    }
```

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



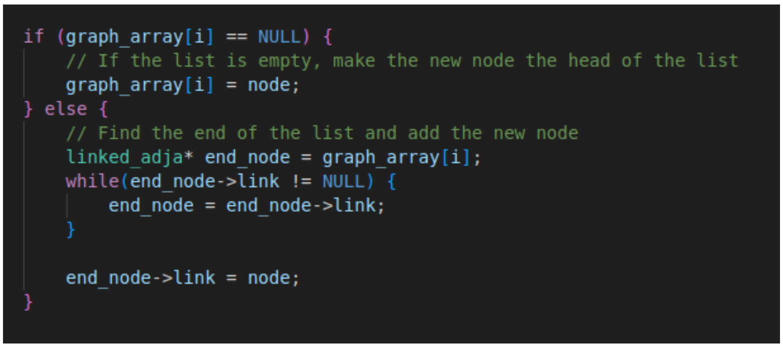
```c
linked_adja** graph_array = (linked_adja**)malloc(v_num * sizeof(linked_adja*));
```

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



```c
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;
```

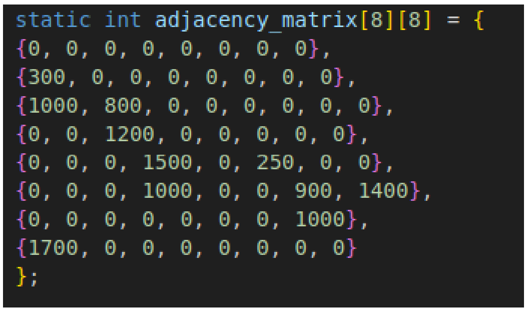
list가 empty인지 아닌지를 확인해서 element를 차례로 넣어준다.



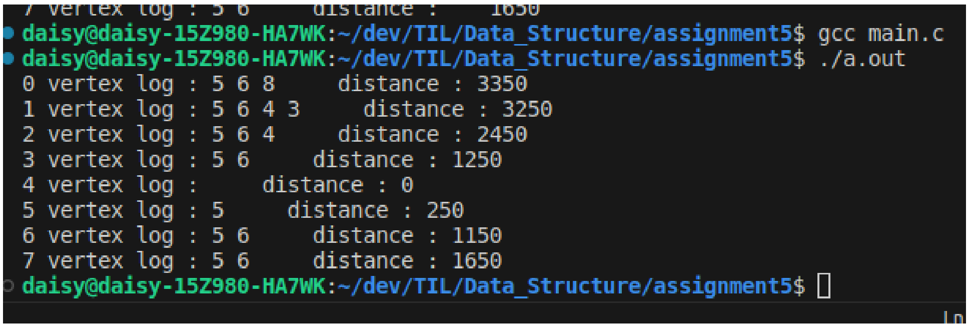
```c
if (graph_array[i] == NULL) {
    // If the list is empty, make the new node the head of the list
    graph_array[i] = node;
} else {
    // Find the end of the list and add the new node
    linked_adja* end_node = graph_array[i];
    while(end_node->link != NULL) {
        end_node = end_node->link;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

    end_node->link = node;
}
```

result

1)

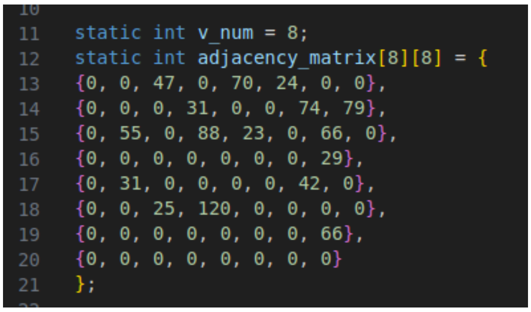


```
static int adjacency_matrix[8][8] = {
{0, 0, 0, 0, 0, 0, 0, 0},
{300, 0, 0, 0, 0, 0, 0, 0},
{1000, 800, 0, 0, 0, 0, 0, 0},
{0, 0, 1200, 0, 0, 0, 0, 0},
{0, 0, 0, 1500, 0, 250, 0, 0},
{0, 0, 0, 1000, 0, 0, 900, 1400},
{0, 0, 0, 0, 0, 0, 0, 1000},
{1700, 0, 0, 0, 0, 0, 0, 0}
};
```

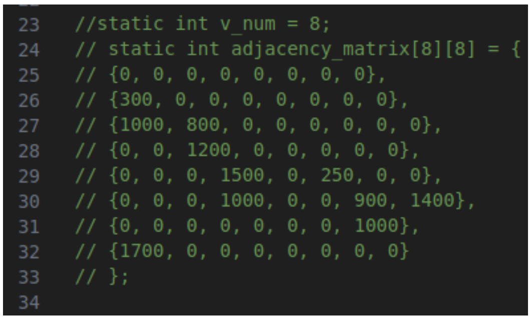


```
7 vertex log : 5 6      distance :    1650
daisy@daisy-15Z980-HA7WK:~/dev/TIL/Data_Structure/assignment5$ gcc main.c
daisy@daisy-15Z980-HA7WK:~/dev/TIL/Data_Structure/assignment5$ ./a.out
0 vertex log : 5 6 8     distance : 3350
1 vertex log : 5 6 4 3     distance : 3250
2 vertex log : 5 6 4    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

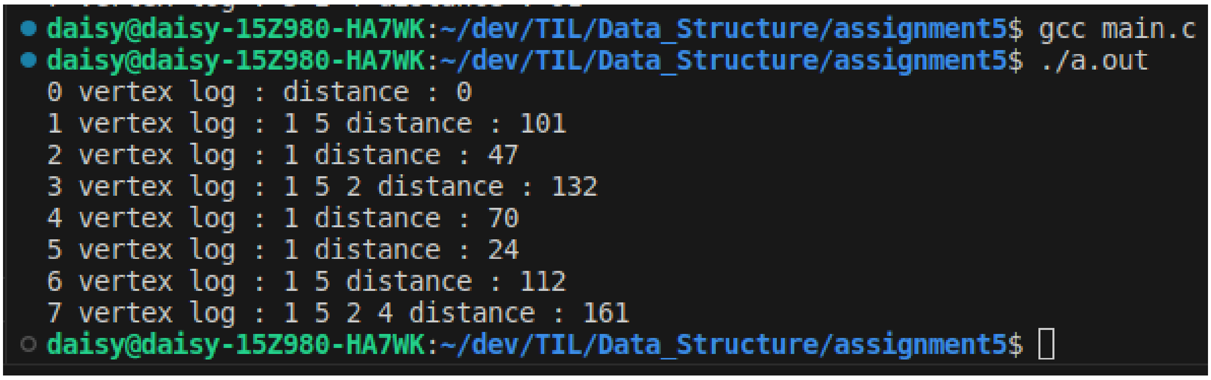
2)
start vertex 0



```
static int v_num = 8;
static int adjacency_matrix[8][8] = {
{0, 0, 47, 0, 70, 24, 0, 0},
{0, 0, 0, 31, 0, 0, 74, 79},
{0, 55, 0, 88, 23, 0, 66, 0},
{0, 0, 0, 0, 0, 0, 0, 29},
{0, 31, 0, 0, 0, 0, 42, 0},
{0, 0, 25, 120, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 66},
{0, 0, 0, 0, 0, 0, 0, 0}
};
```



```
//static int v_num = 8;
// 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}
// };
```



```
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$ 
```