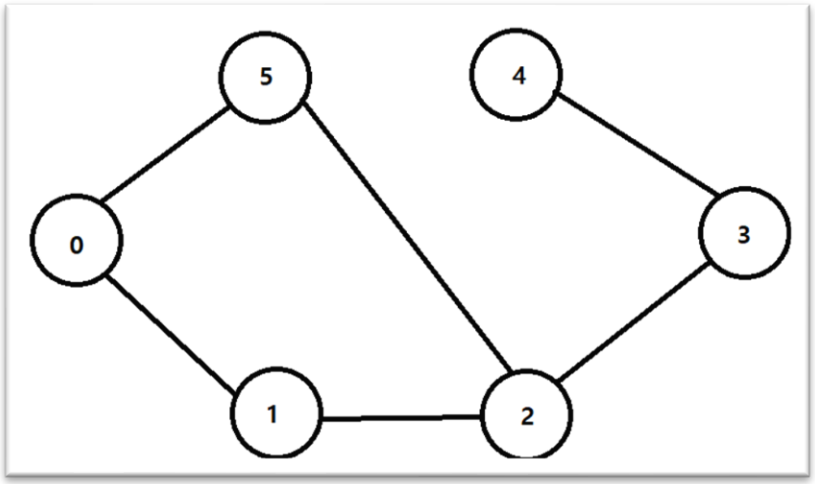
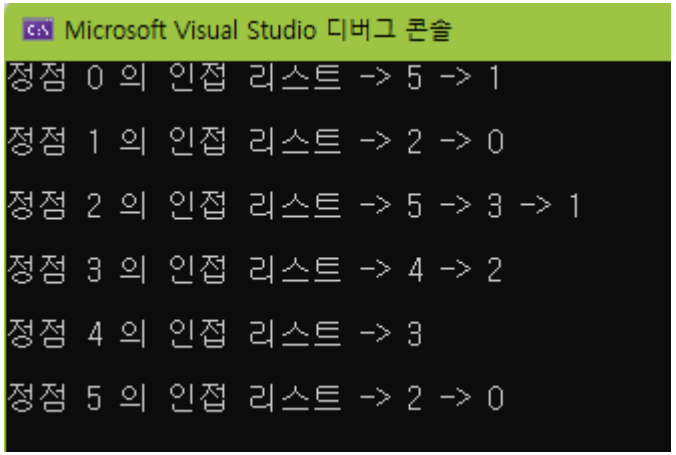
CHAPTER 14

그래프 구현 및 응용

실습 1

문제 1)
인접 리스트 방식으로 아래와 같은 그래프를 구현하고, 각 노드의 인접 리스트를 출력하시오. (무방향 그래프)





(뒷장에 구조체와 함수의 틀이 제시되어있음)

실습 1

문제 1-참고) 다음을 참고하여 코드를 작성하시오.

```
// 노드
typedef struct GraphNode {
    int vertex;                         // 자신의 번호
    struct GraphNode* link;             // 연결된 다른 vertex의 주소
}GraphNode;

// 그래프
typedef struct GraphType {
    int n;                              // edge의 갯수
    GraphNode* adj_list[MAX_VERTICES];  // 인접 리스트
}GraphType;
```

```
// 그래프 초기화
void graph_init(GraphType *g) {
```

```
// vertex 추가
void insert_vertex(GraphType *g, int v) {
```

```
// edge 추가 (v를 u의 인접 리스트에 삽입)
void insert_edge(GraphType *g, int u, int v) {
```

```
// 인접 리스트 출력
void print_adj_list(GraphType* w, int i)
```

실습 1

문제 1 - 정답(1)

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES    50              // 최대 vertex 갯수

// 노드
typedef struct GraphNode {
    int vertex;                         // 자신의 번호
    struct GraphNode* link;             // 연결된 다른 vertex의 주소
}GraphNode;

// 그래프
typedef struct GraphType {
    int n;                              // edge의 갯수
    GraphNode* adj_list[MAX_VERTICES];  // 인접 리스트
}GraphType;

// 그래프 초기화
void graph_init(GraphType *g) {
    int v;
    g->n = 0;                           // 그래프의 edge 갯수 0
    for (v = 0; v < MAX_VERTICES; v++)
        g->adj_list[v] = NULL;          // 인접 리스트 내부의 값 초기화
    return;
}
```

실습 1

문제 1 - 정답(2)

```
// vertex 추가
void insert_vertex(GraphType *g, int v) {
    if ((g->n) + 1 > MAX_VERTICES) {            // 최대 갯수 상황에서는 추가를 금지
        printf("*그래프 : vertex 개수 초과. ₩n");
        return;
    }
    g->n++;
    return;
}

// edge 추가 (v를 u의 인접 리스트에 삽입)
void insert_edge(GraphType *g, int u, int v) {
    GraphNode* node;                                // 임시 노드
    if (u >= g->n || v >= g->n) {                   // 해당 그래프의 vertex의 갯수보다 큰 번호인 경우
        printf("*그래프 : vertex 번호 오류. ₩n");
        return;
    }
    node = (GraphNode *)malloc(sizeof(GraphNode));  // 메모리 공간 할당
    node->vertex = v;                               // 임시 노드에 삽입할 정보 입력 (vertex v)
    node->link = g->adj_list[u];                    // 임시 노드에 삽입할 정보 입력 (기존에 있던 u의 인접 리스트)
    g->adj_list[u] = node;                          // u의 인접 리스트에 임시 노드 정보 추가
    return;
}
```

실습 1

문제 1 - 정답(3)

```
// 인접 리스트 출력 (하나의 vertex만)
void print_adj_list(GraphType* g, int i) {
    GraphNode* p;                                   // 임시 노드
    printf("vertex %d 의 인접 리스트 ", i);

    // i의 인접 리스트 첫 번째 주소 부터, p가 null일 때 까지, p는 인접 리스트의 다음 주소를 받음.
    for (p = g->adj_list[i]; p; p = p->link)
        printf("-> %d ", p->vertex);

    printf("₩n₩n");
    return;
}
```

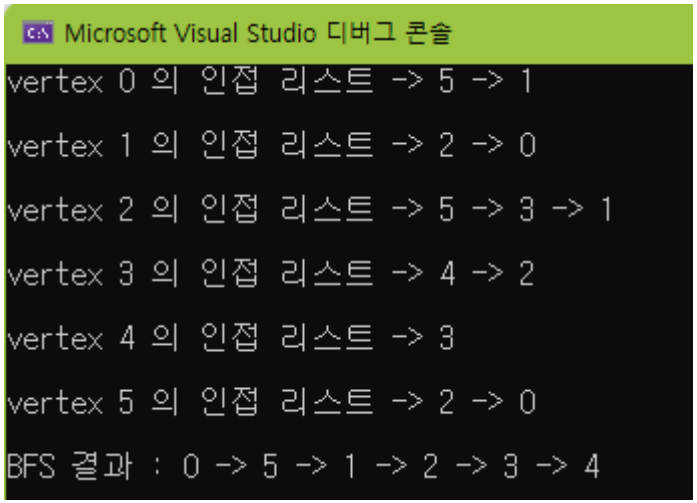
실습 1

문제 1 - 정답(3)

```c
int main() {
    GraphType *g;                                   // 그래프 생성
    g = (GraphType*)malloc(sizeof(GraphType));      // 메모리 할당

    graph_init(g);                                  // 그래프 초기화
    for (int i = 0; i < 6; i++)                     // vertex 추가
        insert_vertex(g, i);

    // edge 추가
    insert_edge(g, 0, 1);
    insert_edge(g, 1, 0);
    insert_edge(g, 0, 5);
    insert_edge(g, 5, 0);
    insert_edge(g, 1, 2);
    insert_edge(g, 2, 1);
    insert_edge(g, 2, 3);
    insert_edge(g, 3, 2);
    insert_edge(g, 3, 4);
    insert_edge(g, 4, 3);
    insert_edge(g, 2, 5);
    insert_edge(g, 5, 2);

    for (int i = 0; i < 6; i++)
        print_adj_list(g, i);                       // 각 vertex의 인접 리스트를 출력

    free(g);                                        // 메모리 할당 해제

    return 0;
}
```

실습 2

문제 2)
문제 1에서 구현한 그래프를 너비 우선 탐색(BFS) 방식으로 순회하는 함수를 제작하시오. (큐 사용)




```
void bfs_list(GraphType* g, int v) {
```

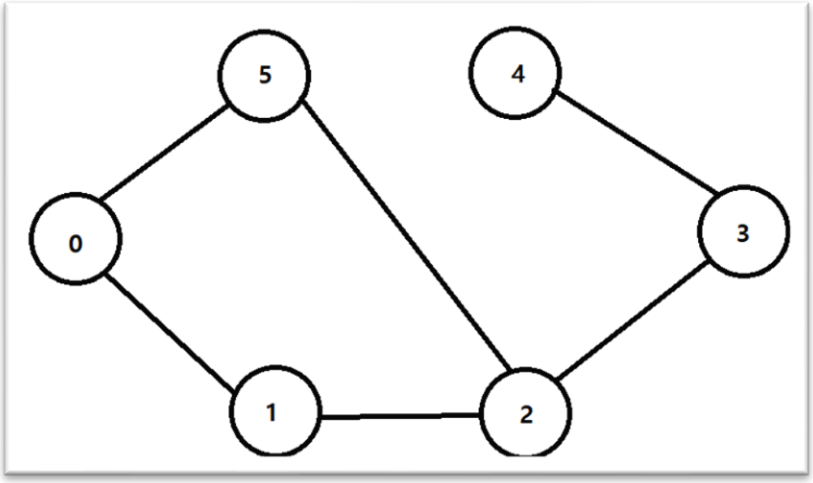
실습 2

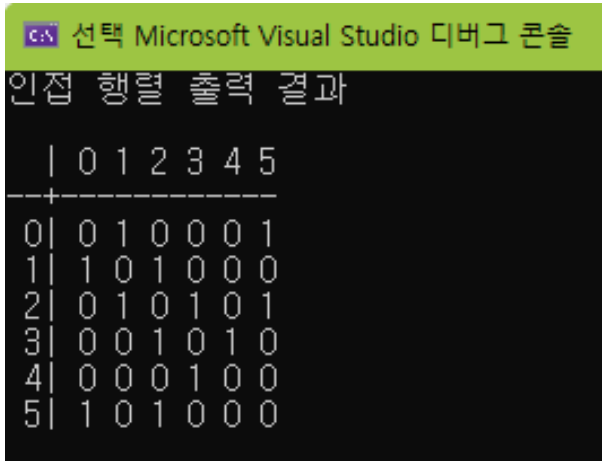
문제 2 - 정답

```
// BFS
void bfs_list(GraphType* g, int v) {
    GraphNode* w;                                   // 임시 노드
    QueueType q;                                    // 큐 생성
    initQueue(&q);                                  // 큐 초기화
    visited[v] = TRUE;                              // 첫 노드 방문 flag TRUE로

    printf("BFS 결과 : ");

    // v 방문
    printf("%d", v);
    enqueue(&q, v);                                 // 첫 노드 큐에 삽입

    while (q.size != 0) {                           // 큐에 원소가 있으면 동작
        v = dequeue(&q);                            // 큐의 원소를 하나 추출

        for (w = g->adj_list[v]; w; w = w->link)    // 추출한 원소의 인접 리스트의 확인. 차례대로 w 노드에 저장.
            if (!visited[w->vertex]) {              // w에 저장된 vertex가 방문되지 않았다면
                visited[w->vertex] = TRUE;          // 방문 flag를 TRUE로
                printf(" -> %d", w->vertex);
                enqueue(&q, w->vertex);             // w에 저장된 vertex 정보를 큐에 삽입 (방문한 vertex의 정보를 큐에 저장)
            }
    }
    printf("₩n");
    return;
}
```

실습 3

문제 3)
인접 행렬 방식으로 아래와 같은 그래프를 구현하고,
인접 행렬을 출력하시오. (무방향 그래프)





(뒷장에 구조체와 함수의 틀이 제시되어있음)

실습 3

문제 3-참고) 다음을 참고하여 코드를 작성하시오.

```c
// 그래프
typedef struct GraphType {
    int n;                                      // vertex의 갯수
    int adj_mat[MAX_VERTICES][MAX_VERTICES];    // 인접 행렬
} GraphType;
```

```c
// 그래프 초기화
void graph_init(GraphType* g)
```

```c
// vertex 추가
void insert_vertex(GraphType* g, int v)
```

```c
// edge 추가
void insert_edge(GraphType* g, int start, int end)
```

```c
// 인접 행렬 출력
void print_adj_mat(GraphType* g)
```

실습 3

문제 3-정답(1)

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

#define MAX_VERTICES 50

// 그래프
typedef struct GraphType {
    int n;                                      // vertex의 갯수
    int adj_mat[MAX_VERTICES][MAX_VERTICES];    // 인접 행렬
} GraphType;


// 그래프 초기화
void graph_init(GraphType* g) {
    int r, c;                                   // row, column
    g->n = 0;
    for (r = 0; r < MAX_VERTICES; r++)
        for (c = 0; c < MAX_VERTICES; c++)
            g->adj_mat[r][c] = 0;               // 인접 행렬 초기화
    return;
}
```

실습 3

문제 3-정답(2)

```
// vertex 추가
void insert_vertex(GraphType* g, int v) {
    if ((g->n) + 1 > MAX_VERTICES) {            // 최대 갯수 상황에서는 추가를 금지
        printf("*그래프 : vertex 개수 초과. ₩n");
        return;
    }
    g->n++;                                     // vertex 추가
    return;
}

// edge 추가
void insert_edge(GraphType* g, int start, int end) {
    if (start >= g->n || end >= g->n) {             // 해당 그래프의 vertex의 갯수보다 큰 번호인 경우
        printf("*그래프 : vertex 번호 오류. ₩n");
        return;
    }

    // 무방향, edge유무 저장
    g->adj_mat[start][end] = 1;
    g->adj_mat[end][start] = 1;
    return;
}
```

실습 3

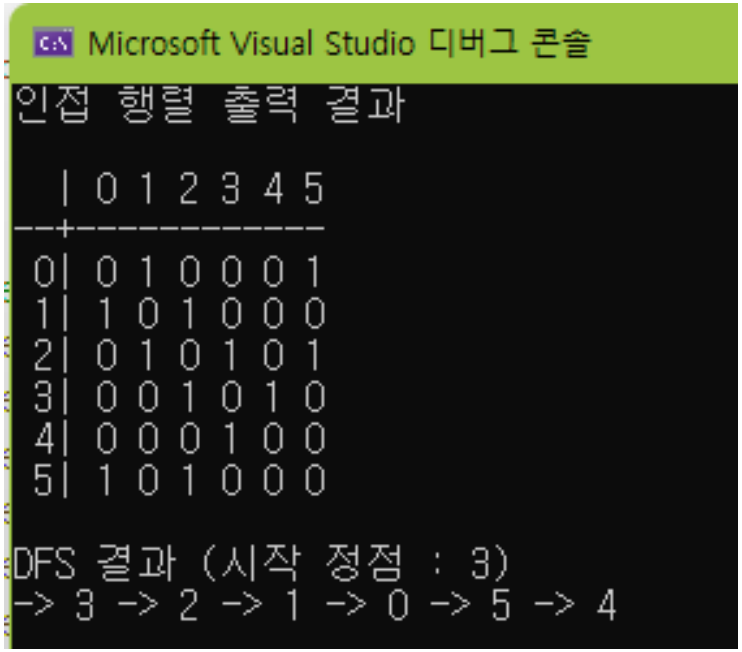
문제 3-정답(3)

```c
// 인접 행렬 출력
void print_adj_mat(GraphType* g) {
    printf("인접 행렬 출력 결과 ₩n₩n");
    printf("  |");

    // 디자인
    for (int i = 0; i< g->n; i++)
            printf("%2d", i);
    printf("₩n--");
    for (int i = 0; i < g->n; i++) {
        if (i == 0) printf("+-");
        else        printf("--");
    }
    printf("-₩n");

    // 인접 행렬 출력
    for (int i = 0; i < g->n; i++) {
        printf("%2d|", i);
        for (int j = 0; j < g->n; j++)
            printf("%2d", g->adj_mat[i][j]);
        printf("₩n");
    }
    return;
}
```

```c
int main() {
    GraphType* g;                                   // 그래프 생성
    g = (GraphType*)malloc(sizeof(GraphType));      // 메모리 할당

    graph_init(g);                                  // 그래프 초기화
    for (int i = 0; i < 6; i++)                     // vertex 추가
        insert_vertex(g, i);

    // edge 추가
    insert_edge(g, 0, 1);
    insert_edge(g, 0, 5);
    insert_edge(g, 1, 2);
    insert_edge(g, 2, 3);
    insert_edge(g, 3, 4);
    insert_edge(g, 2, 5);

    print_adj_mat(g);                               // 인접 행렬 출력

    free(g);                                        // 메모리 할당 해제

    return 0;
}
```

실습 4

문제 4)
문제 3에서 구현한 그래프를 깊이 우선 탐색(DFS) 방식으로 순회하는 함수를 제작하시오. (스택 사용)




```
void dfs_mat(GraphType* g, int v) {
```

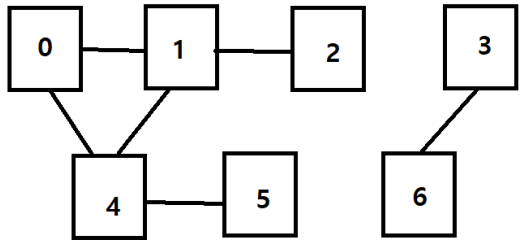
실습 4

문제 4 - 정답

```
// DFS
void dfs_mat(GraphType* g, int v) {
    StackType s;                                    // 스택 생성
    initStack(&s);                                  // 스택 초기화
    printf("DFS 결과 (시작 정점 : %d) ₩n", v);

    push(&s, v);                                    // 스택에 초기 방문 노드 push

    while (!IsEmpty(&s) && !IsAllVisit(g)) {        // 스택이 비어 있지 않고, 노드를 모두 방문한 것이 아니면
        v = pop(&s);                                // 스택의 원소 pop

        if (visited[v])                             // 해당 노드를 방문했다면, 해당 노드는 건너뜀
            continue;

        visited[v] = 1;                             // 방문하지 않은 노드 방문
        printf("-> %d ", v);

        for (int w = g->n - 1; w >= 0; w--) {       // 인접 노드 탐색을 위함
            if (g->adj_mat[v][w] && !visited[w])// 방문하지 않은 인접 노드이면
                push(&s, w);                        // 스택에 push
        }
    }
    return;
}
```

실습 5

문제 5)
7대의 컴퓨터가 아래 그림과 같이 네트워크 상에서 연결되어 있다고 하자. 0번 컴퓨터가 웜 바이러스에 걸리면 웜 바이러스는 1번과 4번 컴퓨터를 거쳐 2번과 5번 컴퓨터까지 전파되어 1, 2, 4, 5 네 대의 컴퓨터는 웜 바이러스에 걸리게 된다. 하지만 3번과 6번 컴퓨터는 1번 컴퓨터와 네트워크상에서 연결되어 있지 않기 때문에 영향을 받지 않는다.



어느 날, 0번 컴퓨터가 웜 바이러스에 걸렸다면, 0번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 출력하는 프로그램을 작성하시오. (0번 컴퓨터는 제외)

실습 5

문제 5-예시)

(입력, 프로그램 내부)

```
// edge 추가
insert_edge(g, 0, 1);
insert_edge(g, 0, 4);
insert_edge(g, 1, 2);
insert_edge(g, 4, 5);
insert_edge(g, 3, 6);

insert_edge(g, 1, 0);
insert_edge(g, 4, 0);
insert_edge(g, 2, 1);
insert_edge(g, 5, 4);
insert_edge(g, 6, 3);
```

출력(콘솔)

```
감염된 컴퓨터의 갯수 : 4
```

실습 5

문제 5 – 정답
해당 해답에서는 문제 2번에 사용되었던 프로그램을 활용하였음.
빨간색 테두리 → 변경점

```
// BFS
int bfs_list(GraphType* g, int v) {
    GraphNode* w;                                   // 임시 노드
    QueueType q;                                    // 큐 생성
    initQueue(&q);                                  // 큐 초기화
    visited[v] = TRUE;                              // 첫 노드 방문 flag TRUE로
    int num = 0;                                    // 방문 노드 확인

    printf("BFS 결과 : ");

    // v 방문
    printf("%d", v);
    enqueue(&q, v);                                 // 첫 노드 큐에 삽입

    while (q.size != 0) {                           // 큐에 원소가 있으면 동작
        v = dequeue(&q);                            // 큐의 원소를 하나 추출

        for (w = g->adj_list[v]; w; w = w->link)    // 추출한 원소의 인접 리스트의 확인. 차례대로 w 노드에 저장.
            if (!visited[w->vertex]) {              // w에 저장된 vertex가 방문되지 않았다면
                visited[w->vertex] = TRUE;          // 방문 flag를 TRUE로
                printf(" -> %d", w->vertex);
                enqueue(&q, w->vertex);             // w에 저장된 vertex 정보를 큐에 삽입 (방문한 vertex의 정보를 큐에 저장)
                num++;
            }
    }
    printf("₩n");
    return num;
}
```

실습 5

문제 5 – 정답
해당 해답에서는 문제 2번에 사용되었던 프로그램을 활용하였음.
빨간색 테두리 → 변경점

```
int main() {
    GraphType* g;                                   // 그래프 생성
    g = (GraphType*)malloc(sizeof(GraphType));      // 메모리 할당

    graph_init(g);                                  // 그래프 초기화
    for (int i = 1; i <= 7; i++)                        // vertex 추가
        insert_vertex(g, i);

    // edge 추가
    insert_edge(g, 0, 1);
    insert_edge(g, 0, 4);
    insert_edge(g, 1, 2);
    insert_edge(g, 4, 5);
    insert_edge(g, 3, 6);

    insert_edge(g, 1, 0);
    insert_edge(g, 4, 0);
    insert_edge(g, 2, 1);
    insert_edge(g, 5, 4);
    insert_edge(g, 6, 3);

    printf("감염된 컴퓨터의 갯수 : %d Wn", bfs_list(g, 0));

    free(g);                                        // 메모리 할당 해제

    return 0;
}
```

실습 6

문제 6)
N * M 크기의 배열로 표현되는 미로가 있다.

1	0	1	1	1	1
1	0	1	0	1	0
1	0	1	0	1	1
1	1	1	0	1	1

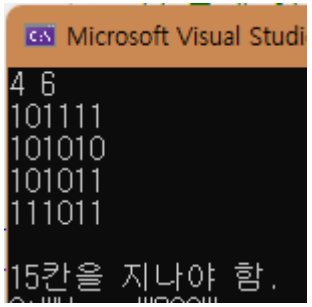
1: 이동할 수 있는 칸
0: 이동할 수 없는 칸

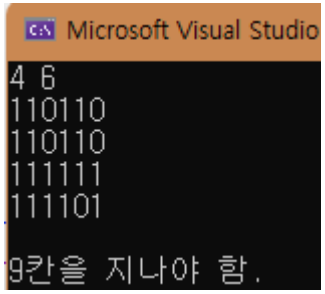
이러한 미로가 주어졌을 때, (1, 1)에서 출발하여 (N, M)의 위치로 이동할 때 지나야 하는 최소의 칸 수를 구하는 프로그램을 작성하시오. (대각선 이동 X)

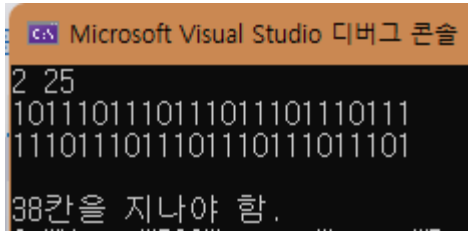
힌트 : BFS를 이용한다.

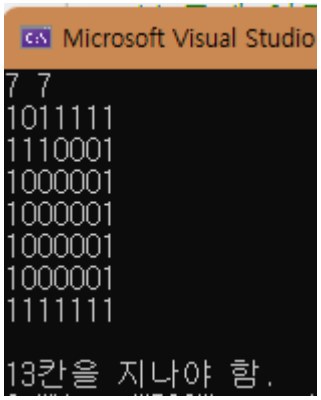
실습 6

문제 6) 입출력 예시









(입력)
첫째 줄 : N M
둘째 줄~ : 미로

(출력)
*(숫자)*칸을 지나야 함.

실습 6

문제 6 – 정답(1)

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#define ROW 100
#define COL 100

int graph[ROW][COL] = { 0, };   // 미로 저장
int q[1001][2] = { 0, };        // 현재 방문한 좌표(x, y) 저장

// 상하좌우 이동을 위한 수
int dx[4] = { -1, 1, 0, 0 };
int dy[4] = { 0, 0, -1, 1 };

int n = 0, m = 0;               // 미로 크기
```

실습 6

문제 6 – 정답(2)

```
int bfs() {
    // 큐 front, rear
    int front = 0, rear = 0;

    // 큐에 처음 (1, 1) 좌표 삽입
    q[front][0] = 1;
    q[front][1] = 1;
    rear++;

    // 큐가 빌 때 까지
    while (front < rear) {
        int x = q[front][0]; // x 좌표
        int y = q[front][1]; // y 좌표
        front++;             // front 이동 (dequeue 역할)

        // 인접 칸(상하좌우) 이동 (인접 정점 탐색)
        for (int i = 0; i < 4; i++) {

            // 인접 정점 좌표
            int nx = x + dx[i];
            int ny = y + dy[i];

            // 범위를 벗어나는 경우, 이동X
            if (nx < 1 || ny < 1 || nx > n || ny > m)
                continue;

            // 길이 아닌 경우, 이동 X
            if (graph[nx][ny] != 1)
                continue;

            // 이전 칸에서 이동한 칸 수 + 1
            graph[nx][ny] = graph[x][y] + 1;        // 이동한 칸에 이동한 칸 수를 갱신

            // 큐에 인접 정점 좌표(nx, ny) 삽입 (enqueue 역할)
            q[rear][0] = nx;
            q[rear][1] = ny;
            rear++;
        }
    }
    return graph[n][m];
}
```

실습 6

문제 6 – 정답(3)

```
int main()
{
    scanf("%d %d", &n, &m);

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            scanf("%1d", &graph[i][j]);
        }
    }

    int ans = bfs();
    printf("₩n%d칸을 지나야 함.", ans);

    return 0;
}
```

질문 및 정리