데이터 구조 10장 실습과제

20223100 박신조

10.1 adj_mat.c

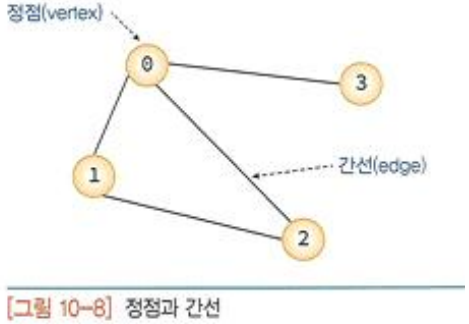
```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_VERTICES 50
typedef struct GraphType {
    int n;
    int adj_mat[MAX_VERTICES][MAX_VERTICES];
} GraphType;


void init(GraphType* g)
{
    int r, c;
    g->n = 0;
    for (r = 0; r < MAX_VERTICES; r++)
        for (c = 0; c < MAX_VERTICES; c++)
            g->adj_mat[r][c] = 0;
}

void insert_vertex(GraphType* g, int v)
{
    if (((g->n) + 1) > MAX_VERTICES) {
        fprintf(stderr, "그래프: 정점의 개수 초과");
        return;
    }
    g->n++;
}

void insert_edge(GraphType* g, int start, int end)
{
    if (start >= g->n || end >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    g->adj_mat[start][end] = 1;
    g->adj_mat[end][start] = 1;
}

void print_adj_mat(GraphType* g)
{
    for (int i = 0; i < g->n; i++) {
        for (int j = 0; j < g->n; j++) {
            printf("%2d ", g->adj_mat[i][j]);
        }
        printf("\n");
    }
}

void main()
{
    GraphType* g;
    g = (GraphType*)malloc(sizeof(GraphType));
    init(g);
    for (int i = 0;i < 4;i++)

        insert_vertex(g, i);
    insert_edge(g, 0, 1);
    insert_edge(g, 0, 2);
    insert_edge(g, 0, 3);
    insert_edge(g, 1, 2);
    insert_edge(g, 2, 3);
    print_adj_mat(g);

    free(g);
}
```

```
 0  1  1  1
 1  0  1  0
 1  1  0  1
 1  0  1  0
```

인접 행렬을 사용하여 그래프를 구현한 코드

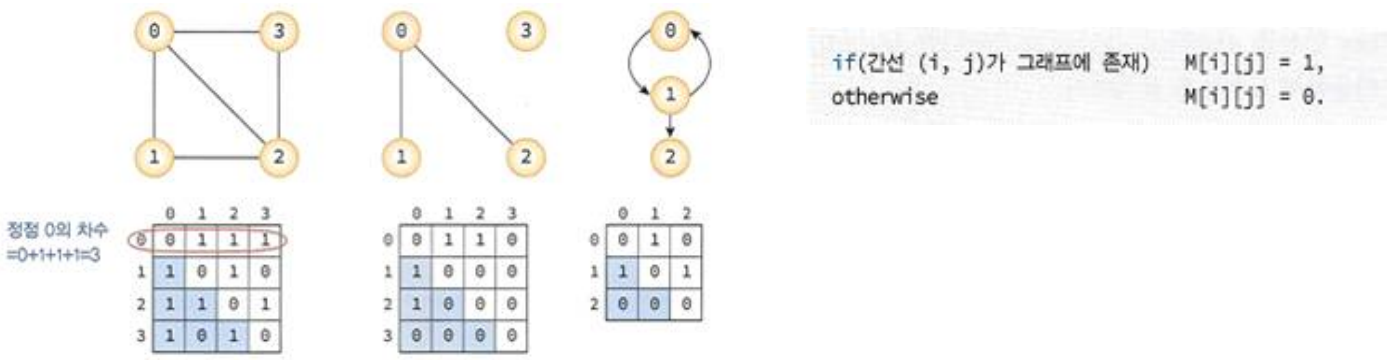
그래프 : 정점(vertex)와 간선(edge)들의 유한 집합

```
V(G1)= { 0, 1, 2, 3 }
E(G1)= { (0, 1), (0, 2), (0, 3), (1, 2) }
```



[그림 10-8] 정점과 간선

인접 행렬 : 2차원 배열을 사용하여 그래프를 표현



#define MAX_VERTICES 50

MAX_VERTICES을 50으로 지정.(최대 정점의 수)

typedef struct GraphType { . . . } GraphType;

그래프의 인접배열을 구현하는 구조체
멤버 : 정점 개수 n, 인접 행렬 adj_mat
adj_mat[i][j]는 정점 i와 j 사이에 간선이 있는지(1 또는 0)를 나타냄

void init(GraphType* g)

그래프의 인접행렬을 초기화하는 함수
g->n = 0으로 정점 수를 초기화
인접 행렬 전체를 0으로 초기화

void insert_vertex(GraphType* g, int v)

정점을 추가하는 함수
g->n을 1 증가시키며, 최대 정점 수를 초과하면 오류 메시지를 출력

void insert_edge(GraphType* g, int start, int end)

간선을 추가하는 함수입니다.
주어진 두 정점 사이에 간선을 추가
adj_mat[start][end]과 adj_mat[end][start]를 1로 설정

void print_adj_mat(GraphType* g)

현재 그래프의 인접 행렬을 출력하는 함수
정점의 수 n만큼 이중 for문으로 인접행렬 출력

void main()

그래프를 동적으로 할당.
insert_vertex()함수를 호출해 정점을 4개로 설정 후 insert_edge()함수로 간선 설정
현재 그래프를 출력
동적으로 할당받은 그래프 반납

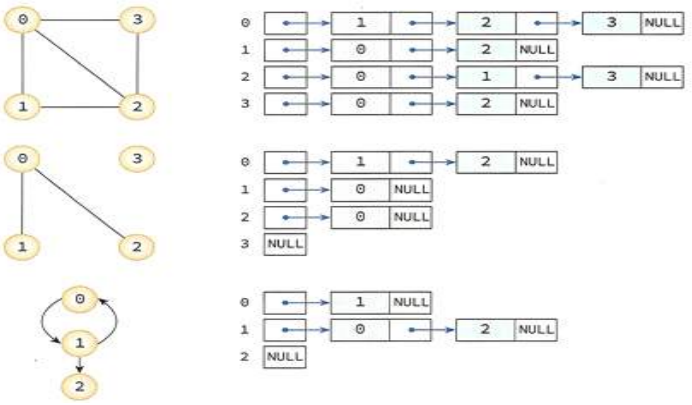
10.2 adj_list.c

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

#define MAX_VERTICES 50
typedef struct GraphNode
{
    int vertex;
    struct GraphNode* link;
} GraphNode;

typedef struct GraphType {
    int n;
    GraphNode* adj_list[MAX_VERTICES];
} GraphType;

void init(GraphType* g)
{
    int v;
    g->n = 0;
    for (v = 0; v < MAX_VERTICES; v++)
        g->adj_list[v] = NULL;
}

void insert_vertex(GraphType* g, int v)
{
    if (((g->n) + 1) > MAX_VERTICES) {
        fprintf(stderr, "그래프: 정점의 개수 초과");
        return;
    }
    g->n++;
}
void insert_edge(GraphType* g, int u, int v)
{
    GraphNode* node;
    if (u >= g->n || v >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    node = (GraphNode*)malloc(sizeof(GraphNode));
    node->vertex = v;
    node->link = g->adj_list[u];
    g->adj_list[u] = node;
}

void print_adj_list(GraphType* g)
{
    for (int i = 0; i < g->n; i++) {
        GraphNode* p = g->adj_list[i];
        printf("정점 %d의 인접 리스트 ", i);
        while (p != NULL) {
            printf("-> %d ", p->vertex);
            p = p->link;
        }
        printf("\n");
    }
}

int main()
{
    GraphType* g;
    g = (GraphType*)malloc(sizeof(GraphType));
    init(g);
    for (int i = 0;i < 4;i++)

        insert_vertex(g, i);
    insert_edge(g, 0, 1);
    insert_edge(g, 1, 0);
    insert_edge(g, 0, 2);
    insert_edge(g, 2, 0);
    insert_edge(g, 0, 3);
    insert_edge(g, 3, 0);
    insert_edge(g, 1, 2);
    insert_edge(g, 2, 1);
    insert_edge(g, 2, 3);
    insert_edge(g, 3, 2);
    print_adj_list(g);
    free(g);
    return 0;
}
```

```
정점 0의 인접 리스트 -> 3 -> 2 -> 1
정점 1의 인접 리스트 -> 2 -> 0
정점 2의 인접 리스트 -> 3 -> 1 -> 0
정점 3의 인접 리스트 -> 2 -> 0
```

인접 리스트를 사용하여 그래프를 구현한 코드

인접 리스트: 연결 리스트를 사용하는 그래프를 표현



typedef struct GraphNode{ . . . } GraphNode;

그래프를 연결리스트로 구현하는 구조체
멤버 : 노드의 번호 vertex, 다음 노드 link

typedef struct GraphType { . . . } GraphType;

정점과 간선을 표현하는 구조체
멤버 : 정점 수 n, 인접 리스트 포인터 배열 adj_list

void init(GraphType* g)

그래프의 인접리스트, 정점을 초기하는 함수
정점 수를 0으로 초기화.
각 인접 리스트를 NULL로 초기화.

void insert_vertex(GraphType* g, int v)

정점을 추가하는 함수
g->n을 1 증가시키며, 최대 정점 수를 초과하면 오류 메시지를 출력

void insert_edge(GraphType* g, int u, int v)

간선을 추가하는 함수
정점 u에서 v로의 간선을 인접 리스트에 추가.
새로운 GraphNode를 생성해서 adj_list[u]의 앞에 삽입

void print_adj_list(GraphType* g)

현재 그래프의 인접리스트를 출력하는 함수
각 정점마다 연결된 정점들을 출력.
리스트가 끝날 때까지 link를 따라 순회.

int main()

그래프를 동적으로 할당.
insert_vertex()함수를 호출해 정점을 4개로 설정 후 insert_edge() 함수로 간선 설정(무방향 그래프라 두 번씩 호출)
print_adj_list()함수를 호출하여 현재 그래프의 인접 리스트을 출력
이후 할당받은 그래프 메모리 반납

10.3 인접 배열로 표현된 그래프에 대한 깊이우선탐색 프로그램(인접 행렬)

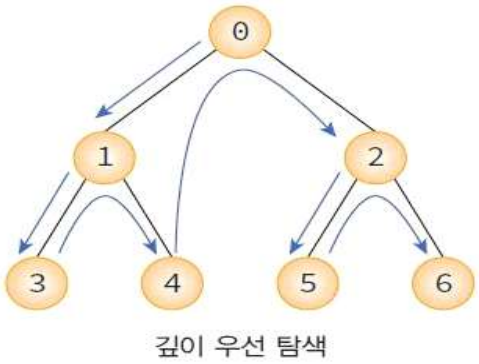
```c
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0
#define MAX_VERTICES 50
typedef struct GraphType {
    int n;
    int adj_mat[MAX_VERTICES][MAX_VERTICES];
} GraphType;

int visited[MAX_VERTICES];

void init(GraphType* g)
{
    int r, c;
    g->n = 0;
    for (r = 0; r < MAX_VERTICES; r++)
        for (c = 0; c < MAX_VERTICES; c++)
            g->adj_mat[r][c] = 0;
}

void insert_vertex(GraphType* g, int v)
{
    if (((g->n) + 1) > MAX_VERTICES) {
        fprintf(stderr, "그래프: 정점의 개수 초과");
        return;
    }
    g->n++;
}

void insert_edge(GraphType* g, int start, int end)
{
    if (start >= g->n || end >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    g->adj_mat[start][end] = 1;
    g->adj_mat[end][start] = 1;
}
void dfs_mat(GraphType* g, int v)
{
    int w;
    visited[v] = TRUE;
    printf("정점 %d -> ", v);
    for (w = 0; w < g->n; w++)
        if (g->adj_mat[v][w] && !visited[w])
            dfs_mat(g, w);
}
int main(void)
{
    GraphType* g;
    g = (GraphType*)malloc(sizeof(GraphType));
    init(g);
    for (int i = 0; i < 4; i++)
        insert_vertex(g, i);
    insert_edge(g, 0, 1);
    insert_edge(g, 0, 2);
    insert_edge(g, 0, 3);
    insert_edge(g, 1, 2);
    insert_edge(g, 2, 3);

    printf("깊이 우선 탐색₩n");
    dfs_mat(g, 0);
    printf("₩n");
    free(g);
    return 0;
}
```

```
깊이 우선 탐색
정점 0 -> 정점 1 -> 정점 2 -> 정점 3 ->
```

인접 행렬 방식으로 그래프를 구현하고

깊이 우선 탐색(DFS, Depth-First Search) 을 수행하는 코드

깊이우선 탐색이란?
그래프에서 깊은 부분을 우선 적으로 탐색하는 알고리즘



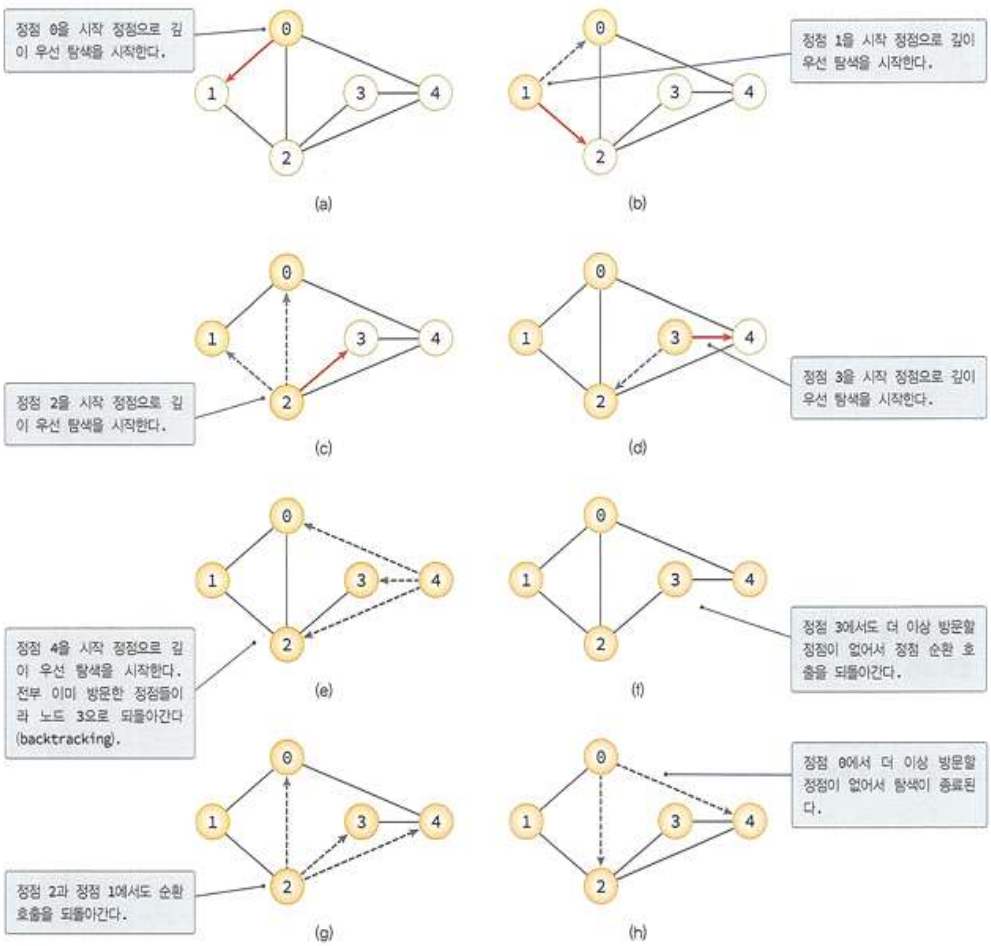
깊이 우선 탐색

[그림 10-18] DFS와 BFS

순서 0 -> 1 -> 3 -> 4 -> 2 -> 5 -> 6

깊이우선탐색 의사코드

```
depth_first_search(v):
    v를 방문되었다고 표시;
        for all u ∈ (v에 인접한 정점) do
            if (u가 아직 방문되지 않았으면)
                then depth_first_search(u)
```



#define TRUE 1

#define FALSE 0

True를 1로, FALSE를 0으로 지정(bool형으로 사용하기 위해)

typedef struct GraphType { . . . } GraphType;

그래프의 인접배열을 구현하는 구조체
멤버 : 정점 개수 n, 인접 행렬 adj_mat
adj_mat[i][j]는 정점 i와 j 사이에 간선이 있는지(1 또는 0)를 나타냄

int visited[MAX_VERTICES]

DFS 중 방문한 정점을 추적하는 배열

void init(GraphType* g)

그래프의 인접행렬을 초기화하는 함수
g->n = 0으로 정점 수를 초기화
인접 행렬 전체를 0으로 초기화

void insert_vertex(GraphType* g, int v)

정점을 추가하는 함수
g->n을 1 증가시키며, 최대 정점 수를 초과하면 오류 메시지를 출력

void insert_edge(GraphType* g, int start, int end)

간선을 추가하는 함수입니다.
주어진 두 정점 사이에 간선을 추가
adj_mat[start][end]과 adj_mat[end][start]를 1로 설정

void dfs_mat(GraphType* g, int v)

그래프의 인접배열을 깊이우선탐색 하는 함수
현재 정점 v를 방문하고 출력.
visited[v] = TRUE로 방문 표시.
인접한 정점 중 방문하지 않은 정점이 있으면 재귀 호출.

int main(void)

그래프 동적으로 할당 및 초기화
void insert_vertex() 함수를 호출하여 정점 4개 설정
void insert_edge() 함수를 호출하여 각 정점 간 간선 설정
깊이우선 탐색(DFS) 실시
0번 정점부터 차례대로 연결된 정점들 탐색
이후 할당받은 그래프 메모리 반납

10.4 인접 배열로 표현된 그래프에 대한 깊이우선탐색 프로그램(인접 리스트)

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

#define MAX_VERTICES 50
typedef struct GraphNode
{
    int vertex;
    struct GraphNode* link;
} GraphNode;

typedef struct GraphType {
    int n;
    GraphNode* adj_list[MAX_VERTICES];
} GraphType;

int visited[MAX_VERTICES];

void init(GraphType* g)
{
    int v;
    g->n = 0;
    for (v = 0; v < MAX_VERTICES; v++)
        g->adj_list[v] = NULL;
}

void insert_vertex(GraphType* g, int v)
{
    if (((g->n) + 1) > MAX_VERTICES) {
        fprintf(stderr, "그래프: 정점의 개수 초과");
        return;
    }
    g->n++;
}
void insert_edge(GraphType* g, int u, int v)
{
    GraphNode* node;
    if (u >= g->n || v >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    node = (GraphNode*)malloc(sizeof(GraphNode));
    node->vertex = v;
    node->link = g->adj_list[u];
    g->adj_list[u] = node;
}

void dfs_list(GraphType* g, int v)
{
    GraphNode* w;
    visited[v] = 1;
    printf("정점 %d -> ", v);
    for (w = g->adj_list[v]; w; w = w->link)
        if (!visited[w->vertex])
            dfs_list(g, w->vertex);
}
int main()
{
    GraphType* g;
    g = (GraphType*)malloc(sizeof(GraphType));
    init(g);
    for (int i = 0;i < 4;i++)
        insert_vertex(g, i);

    insert_edge(g, 0, 1);
    insert_edge(g, 0, 2);
    insert_edge(g, 0, 3);
    insert_edge(g, 1, 2);
    insert_edge(g, 2, 3);

    printf("깊이 우선 탐색₩n");
    dfs_list(g, 0);
    printf("₩n");
    free(g);
    return 0;
}
```

```
깊이 우선 탐색
정점 0 -> 정점 3 -> 정점 2 -> 정점 1 ->
```

인접 리스트 방식으로 그래프를 구현하고
깊이 우선 탐색(DFS, Depth-First Search) 을 수행하는 코드

typedef struct GraphNode{ . . . } GraphNode;

그래프를 연결리스트로 구현하는 구조체
멤버 : 노드의 번호 vertex, 다음 노드 link

typedef struct GraphType { . . . } GraphType;

정점과 간선을 표현하는 구조체
멤버 : 정점 수 n, 인접 리스트 포인터 배열 adj_list

int visited[MAX_VERTICES]

DFS 중 방문한 정점을 추적하는 배열

void init(GraphType* g)

그래프의 인접리스트, 정점을 초기하는 함수
정점 수를 0으로 초기화.
각 인접 리스트를 NULL로 초기화.

void insert_vertex(GraphType* g, int v)

정점을 추가하는 함수
g->n을 1 증가시키며, 최대 정점 수를 초과하면 오류 메시지를 출력

void insert_edge(GraphType* g, int u, int v)

간선을 추가하는 함수
정점 u에서 v로의 간선을 인접 리스트에 추가.
새로운 GraphNode를 생성해서 adj_list[u]의 앞에 삽입

void dfs_list(GraphType* g, int v)

그래프의 인접리스트를 깊이우선탐색 하는 함수
정점 v를 방문 -> 출력하고 visited[v]를 True로 설정.
정점 v의 인접 리스트를 순회하며, 방문하지 않은 정점에 대해 재귀 호출.
v번째 리스트부터 시작하여 연결된 다른 정점을 모두 방문할때까지 반복

int main(void)

그래프 동적으로 할당 및 초기화
void insert_vertex() 함수를 호출하여 정점 4개 설정
void insert_edge() 함수를 호출하여 각 정점 간 간선 설정
깊이우선 탐색(DFS) 실시
0번 정점부터 차례대로 연결된 정점들 탐색
이후 할당받은 그래프 메모리 반납

10.5 너비 우선 탐색(인접 행렬 표현) 프로그램

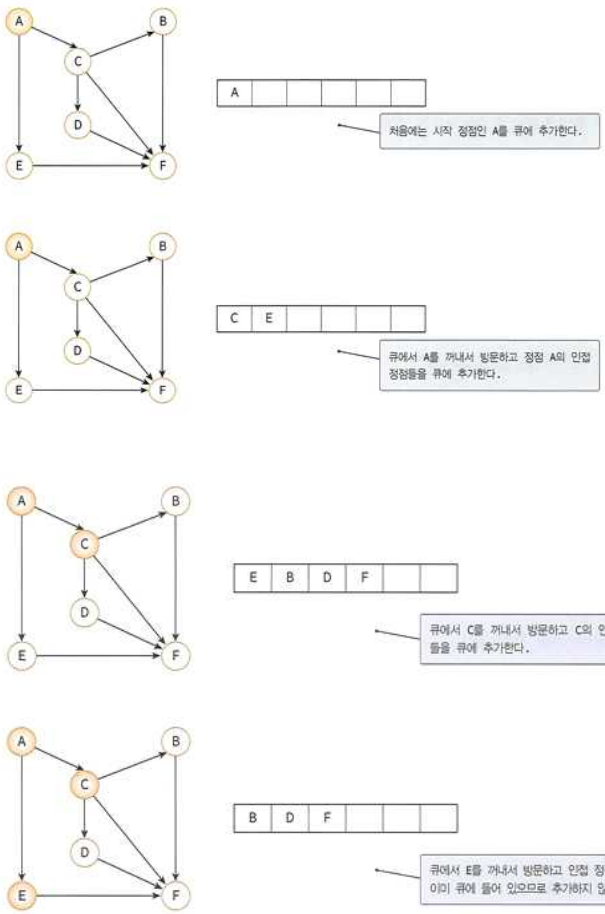
```c
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0
#define MAX_QUEUE_SIZE 10

typedef int element;

typedef struct {
    element  queue[MAX_QUEUE_SIZE];
    int  front, rear;
} QueueType;

void error(char* message)
{
    fprintf(stderr, "%s\n", message);
    exit(1);
}

void queue_init(QueueType* q)
{
    q->front = q->rear = 0;
}

int is_empty(QueueType* q)
{
    return (q->front == q->rear);
}

int is_full(QueueType* q)
{
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}
void enqueue(QueueType* q, element item)
{
    if (is_full(q))
        error("큐가 포화상태입니다");
    q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    q->queue[q->rear] = item;
}

element dequeue(QueueType* q)
{
    if (is_empty(q))
        error("큐가 공백상태입니다");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->queue[q->front];
}

#define MAX_VERTICES 50
typedef struct GraphType {
    int n;
    int adj_mat[MAX_VERTICES][MAX_VERTICES];
} GraphType;
int visited[MAX_VERTICES];

void graph_init(GraphType* g)
{
    int r, c;
    g->n = 0;
    for (r = 0; r < MAX_VERTICES; r++)
        for (c = 0; c < MAX_VERTICES; c++)
            g->adj_mat[r][c] = 0;
}
void insert_vertex(GraphType* g, int v)
{
    if (((g->n) + 1) > MAX_VERTICES) {
        fprintf(stderr, "그래프: 정점의 개수 초과");
        return;
    }
    g->n++;
}

void insert_edge(GraphType* g, int start, int end)
{
    if (start >= g->n || end >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    g->adj_mat[start][end] = 1;
    g->adj_mat[end][start] = 1;
}
```

```c
void bfs_mat(GraphType* g, int v)
{
    int w;
    QueueType q;

    queue_init(&q);
    visited[v] = TRUE;
    printf("%d 방문 -> ", v);
    enqueue(&q, v);
    while (!is_empty(&q)) {
        v = dequeue(&q);
        for (w = 0; w < g->n; w++)
            if (g->adj_mat[v][w] && !visited[w]) {
                visited[w] = TRUE;
                printf("%d 방문 -> ", w);
                enqueue(&q, w);
            }
    }
}
int main(void)
{
    GraphType* g;
    g = (GraphType*)malloc(sizeof(GraphType));
    graph_init(g);
    for (int i = 0; i < 6; i++)
        insert_vertex(g, i);
    insert_edge(g, 0, 2);
    insert_edge(g, 2, 1);
    insert_edge(g, 2, 3);
    insert_edge(g, 0, 4);
    insert_edge(g, 4, 5);
    insert_edge(g, 1, 5);

    printf("너비 우선 탐색₩n");
    bfs_mat(g, 0);
    printf("₩n");
    free(g);
    return 0;
}
```

```
너비 우선 탐색
0 방문 -> 2 방문 -> 4 방문 -> 1 방문 -> 3 방문 -> 5 방문 ->
```

인접 행렬 방식으로 그래프를 구현하고,
너비 우선 탐색(BFS, Breadth-First Search) 을 수행하는 코드

너비우선탐색이란?
그래프에서 가까운 노드부터 탐색하는 알고리즘

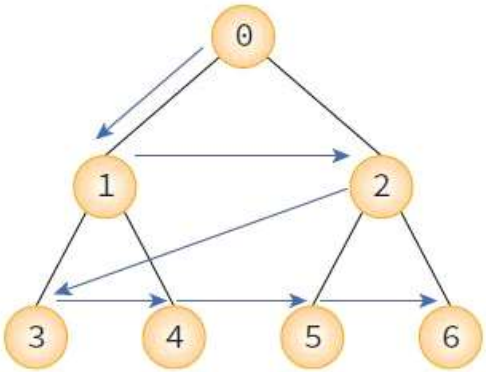


너비우선탐색 의사코드

```
breadth_first_search(v):
    v를 방문되었다고 표시;
    큐 Q에 정점 v를 삽입;
    while (Q가 공택이 아니면) do
        Q에서 정점 w를 삭제;
        for all u ∈ (w에 인접한 정점) do
        if (u가 아직 방문되지 않았으면)
            then u를 큐에 삽입;
            u를 방문되었다고 표시;
```

#define MAX_QUEUE_SIZE 10

MAX_QUEUE_SIZE를 10으로 지정(큐의 최대 크기 10)



너비 우선 탐색

typedef int element;

element의 타입을 int 로 지정

typedef struct { . . . } QueueType

큐를 구현할 구조체
멤버 : 처음 요소 front, 마지막 요소 rear, 정점을 저장할

void error(char* message)

오류 메시지를 출력하고 프로그램을 종료하는 함수
덱이 포화 상태일 경우나 공백일 경우에 사용

void queue_init(QueueType* q)

큐를 초기화하는 함수
front와 rear 인덱스를 0으로 설정하여 초기 상태 설정

int is_empty(QueueType* q)

큐가 공백상태인지 확인하는 함수
front와 rear가 같은 경우 큐가 비어있음을 의미

int is_full(QueueType* q)

큐가 포화상태인지 확인하는 함수
원형 큐에서는 rear의 다음 위치가 front와 같을 때 포화 상태

void enqueue(QueueType* q, element item)

큐에 새로운 데이터를 추가하는 함수
is_full 함수를 통해 큐가 가득 찼는지 확인 후, rear를 1만큼 증가시키고 해당 위치에 데이터를 저장
삽입 후 큐의 요소(배열)를 확인할 수 있도록 print_queue 함수를 호출

element dequeue(QueueType* q)

큐에서 요소를 제거하는 함수
is_empty 함수를 통해 큐에 요소가 있는지 확인 후, front를 1만큼 증가시키고 해당 위치의 인덱스를 반환
배열에서 실제로 데이터를 삭제하지는 않지만, front를 통해 범위를 조정

typedef struct GraphType { . . . } GraphType;

그래프의 인접배열을 구현하는 구조체
멤버 : 정점 개수 n, 인접 행렬 adj_mat
adj_mat[i][j]는 정점 i와 j 사이에 간선이 있는지(1 또는 0)를 나타냄

int visited[MAX_VERTICES]

DFS 중 방문한 정점을 추적하는 배열

void graph_init(GraphType* g)

그래프의 인접행렬을 초기화하는 함수
g->n = 0으로 정점 수를 초기화
인접 행렬 전체를 0으로 초기화

void insert_vertex(GraphType* g, int v)

정점을 추가하는 함수
g->n을 1 증가시키며, 최대 정점 수를 초과하면 오류 메시지를 출력

void insert_edge(GraphType* g, int start, int end)

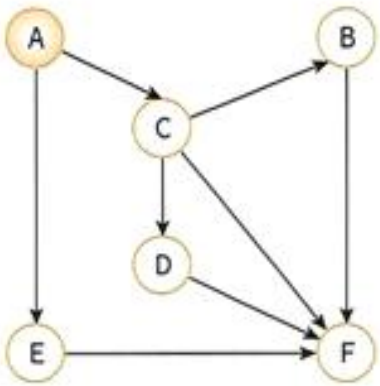
간선을 추가하는 함수입니다.
주어진 두 정점 사이에 간선을 추가
adj_mat[start][end]과 adj_mat[end][start]를 1로 설정

void bfs_mat(GraphType* g, int v)

그래프의 인접 행렬을 너비우선탐색을 하는 함수
정점 v를 방문 -> 출력하고 visited[v]를 True로 설정 큐에 삽입
큐에서 하나씩 꺼내 인접한 정점들을 차례로 탐색
방문하지 않은 정점이면 방문 표시 후 큐에 삽입
인접 행렬을 통해 각 정점의 인접 정점들을 차례로 탐색
v부터 연결된 정점들 차례대로 방문 후 순서대로 정점 번호 출력

int main(void)

그래프 동적으로 할당 및 초기화
void insert_vertex() 함수를 호출하여 정점 6개 설정
void insert_edge() 함수를 호출하여 각 정점 간 간선 설정
너비우선탐색(BFS) 실시
0번 정점부터 차례대로 연결된 정점들 탐색
이후 할당받은 그래프 메모리 반납



10.6 너비 우선 탐색(인접 리스트 표현) 프로그램

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

#define TRUE 1
#define FALSE 0
#define MAX_QUEUE_SIZE 10

typedef int element;
typedef struct {
    element  queue[MAX_QUEUE_SIZE];
    int  front, rear;
} QueueType;

void error(char* message)
{
    fprintf(stderr, "%s\n", message);
    exit(1);
}

void queue_init(QueueType* q)
{
    q->front = q->rear = 0;
}

int is_empty(QueueType* q)
{
    return (q->front == q->rear);
}

int is_full(QueueType* q)
{
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}
void enqueue(QueueType* q, element item)
{
    if (is_full(q))
        error("큐가 포화상태입니다");
    q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    q->queue[q->rear] = item;
}

element dequeue(QueueType* q)
{
    if (is_empty(q))
        error("큐가 공백상태입니다");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->queue[q->front];
}

#define MAX_VERTICES 50

typedef struct GraphNode
{
    int vertex;
    struct GraphNode* link;
} GraphNode;

typedef struct GraphType {
    int n;
    GraphNode* adj_list[MAX_VERTICES];
} GraphType;

int visited[MAX_VERTICES];

void init(GraphType* g)
{
    int v;
    g->n = 0;
    for (v = 0; v < MAX_VERTICES; v++)
        g->adj_list[v] = NULL;
}
void insert_vertex(GraphType* g, int v)
{
    if (((g->n) + 1) > MAX_VERTICES) {
        fprintf(stderr, "그래프: 정점의 개수 초과");
        return;
    }
    g->n++;
}
```

```c
void insert_edge(GraphType* g, int u, int v)
{
    GraphNode* node;
    if (u >= g->n || v >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    node = (GraphNode*)malloc(sizeof(GraphNode));
    node->vertex = v;
    node->link = g->adj_list[u];
    g->adj_list[u] = node;
}

void bfs_list(GraphType* g, int v)
{
    GraphNode* w;
    QueueType q;

    queue_init(&q);
    visited[v] = 1;
    printf("%d 방문 -> ", v);
    enqueue(&q, v);
    while (!is_empty(&q)) {
        v = dequeue(&q);
        for (w = g->adj_list[v]; w; w = w->link)
            if (!visited[w->vertex]) {
                visited[w->vertex] = TRUE;
                printf("%d 방문 -> ", w->vertex);
                enqueue(&q, w->vertex);
            }
    }
}

int main(void)
{
    GraphType* g;
    g = (GraphType*)malloc(sizeof(GraphType));
    init(g);
    for (int i = 0; i < 6; i++)
        insert_vertex(g, i);
    insert_edge(g, 0, 2);
    insert_edge(g, 2, 1);
    insert_edge(g, 2, 3);
    insert_edge(g, 0, 4);
    insert_edge(g, 4, 5);
    insert_edge(g, 1, 5);

    printf("너비 우선 탐색₩n");
    bfs_list(g, 0);
    printf("₩n");
    free(g);
    return 0;
}
```

```
너비 우선 탐색
0 방문 -> 4 방문 -> 2 방문 -> 5 방문 -> 3 방문 -> 1 방문 ->
```

인접 리스트 방식으로 그래프를 구현하고,
너비 우선 탐색(BFS, Breadth-First Search) 을 수행하는 코드

typedef int element;

element의 타입을 int 로 지정

typedef struct { . . . } QueueType

큐를 구현할 구조체
멤버 : 처음 요소 front, 마지막 요소 rear, 정점을 저장할

void error(char* message)

오류 메시지를 출력하고 프로그램을 종료하는 함수
덱이 포화 상태일 경우나 공백일 경우에 사용

void queue_init(QueueType* q)

큐를 초기화하는 함수
front와 rear 인덱스를 0으로 설정하여 초기 상태 설정

int is_empty(QueueType* q)

큐가 공백상태인지 확인하는 함수
front와 rear가 같은 경우 큐가 비어있음을 의미

int is_full(QueueType* q)

큐가 포화상태인지 확인하는 함수
원형 큐에서는 rear의 다음 위치가 front와 같을 때 포화 상태

void enqueue(QueueType* q, element item)

큐에 새로운 데이터를 추가하는 함수
is_full 함수를 통해 큐가 가득 찼는지 확인 후, rear를 1만큼 증가시키고 해당 위치에 데이터를 저장
삽입 후 큐의 요소(배열)를 확인할 수 있도록 print_queue 함수를 호출

element dequeue(QueueType* q)

큐에서 요소를 제거하는 함수
is_empty 함수를 통해 큐에 요소가 있는지 확인 후, front를 1만큼 증가시키고 해당 위치의 인덱스를 반환
배열에서 실제로 데이터를 삭제하지는 않지만, front를 통해 범위를 조정

typedef struct GraphNode{ . . . } GraphNode;

그래프를 연결리스트로 구현하는 구조체
멤버 : 노드의 번호 vertex, 다음 노드 link

typedef struct GraphType { . . . } GraphType;

정점과 간선을 표현하는 구조체
멤버 : 정점 수 n, 인접 리스트 포인터 배열 adj_list

int visited[MAX_VERTICES]

DFS 중 방문한 정점을 추적하는 배열

void graph_init(GraphType* g)

그래프의 인접리스트, 정점을 초기하는 함수
정점 수를 0으로 초기화.
각 인접 리스트를 NULL로 초기화.

void insert_vertex(GraphType* g, int v)

정점을 추가하는 함수
g->n을 1 증가시키며, 최대 정점 수를 초과하면 오류 메시지를 출력

void insert_edge(GraphType* g, int u, int v)

간선을 추가하는 함수
정점 u에서 v로의 간선을 인접 리스트에 추가.
새로운 GraphNode를 생성해서 adj_list[u]의 앞에 삽입

void bfs_list(GraphType* g, int v)

그래프의 인접 리스트를 너비우선탐색을 하는 함수
정점 v를 방문 -> 출력하고 visited[v]를 True로 설정 큐에 삽입

큐에서 하나씩 꺼내 인접한 정점들을 차례로 탐색
방문하지 않은 정점이면 방문 표시 후 큐에 삽입
인접 리스트를 통해 각 정점의 인접 정점들을 차례로 탐색
v부터 연결된 정점들 차례대로 방문 후 순서대로 정점 번호 출력

int main(void)

그래프 동적으로 할당 및 초기화
void insert_vertex() 함수를 호출하여 정점 6개 설정
void insert_edge() 함수를 호출하여 각 정점 간 간선 설정
너비우선탐색(BFS) 실시
0번 정점부터 차례대로 연결된 정점들 탐색
이후 할당받은 그래프 메모리 반납