



# 자료구조 실습

12/02





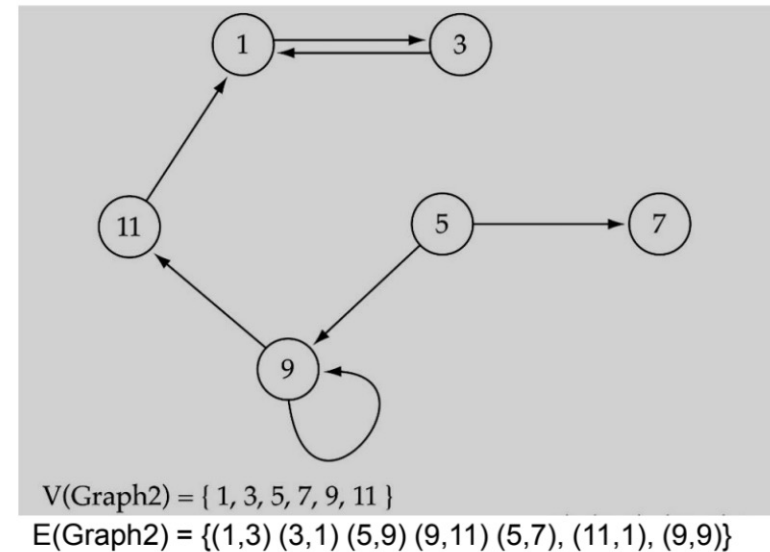
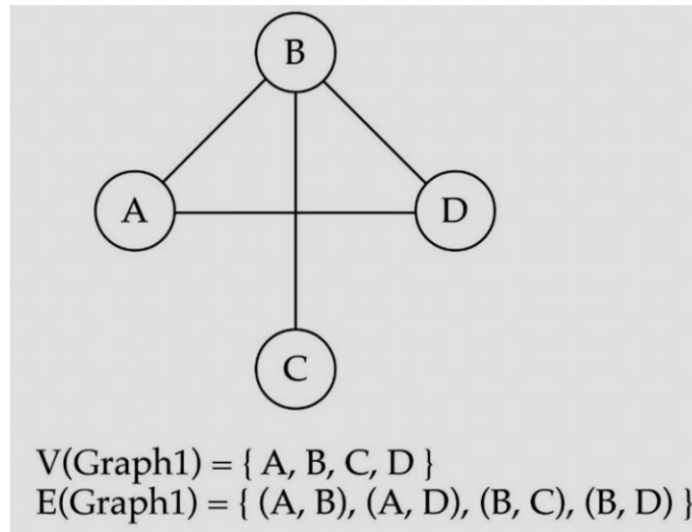
# 목차

- Graph란?
- 인접 행렬로 구현
- 인접 리스트로 구현
  
- 실행 결과



# Graph란?

- 정점(Vertex)과 간선(Edge)
- 방향성 여부 : 무방향(undirected) / 방향(directed)
- 가중치 여부 : 가중치 없는(unweighted) / 가중치 있는(weighted)





# 인접 행렬로 구현

- 정점 수  $\times$  정점 수의 2차원 배열  
정점  $i, j$  연결 여부를 값으로
- 장점
  - 간선 존재 여부 확인  $O(1)$
  - 구현이 단순함
  - 간선이 많은 그래프에 적합
- 단점
  - 정점이 많으면 메모리 사용량이 큼 ( $n^2$  공간)
  - 간선 추가/삭제는 빠르지만  
연결된 정점 전체 탐색은 느릴 수 있음

graph

.numVertices 7  
.vertices

[0]	"Atlanta"	"
[1]	"Austin"	"
[2]	"Chicago"	"
[3]	"Dallas"	"
[4]	"Denver"	"
[5]	"Houston"	"
[6]	"Washington"	"
[7]		
[8]		
[9]		

.edges

[0]	0	0	0	0	0	800	600	•	•	•
[1]	0	0	0	200	0	160	0	•	•	•
[2]	0	0	0	0	1000	0	0	•	•	•
[3]	0	200	900	0	780	0	0	•	•	•
[4]	1400	0	1000	0	0	0	0	•	•	•
[5]	800	0	0	0	0	0	0	•	•	•
[6]	600	0	0	1300	0	0	0	•	•	•
[7]	•	•	•	•	•	•	•	•	•	•
[8]	•	•	•	•	•	•	•	•	•	•
[9]	•	•	•	•	•	•	•	•	•	•

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

(Array positions marked '•' are undefined)



# 인접 리스트로 구현

- 연결된 정점을 단일 연결 리스트로 나열

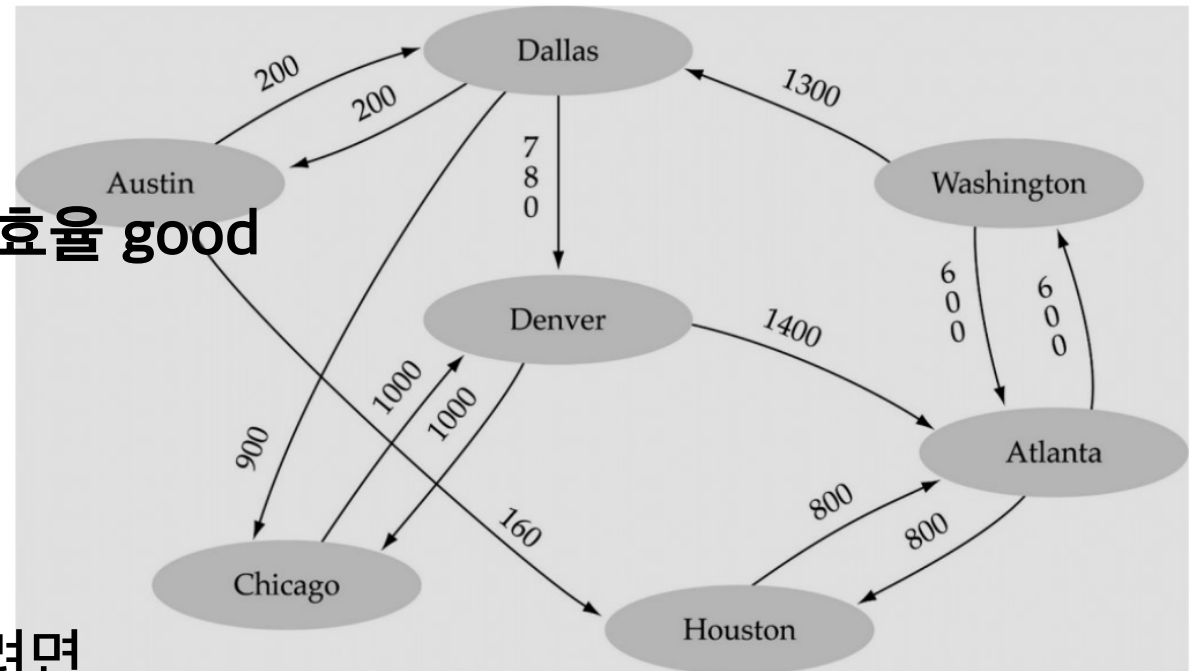
- $2 \rightarrow 0 \rightarrow 1 \rightarrow 3 \rightarrow \text{NULL}$

- 장점

- 필요한 연결만 저장하므로 메모리 효율 good
  - 간선이 적은 그래프에 적합
  - 정점의 이웃을 탐색하는 데 유리 (연결된 노드만 보면 됨)

- 단점

- 두 정점이 연결되어 있는지 확인하려면  
기준으로 잡은 정점 u에 연결된 이웃 정점의 수 만큼 탐색 필요
  - 구현이 행렬보다 복잡함





# 인접 행렬로 구현

```
#define MAX_VERTICES 10 // 그래프에서 사용할 수 있는 최대 정점 개수를 상수로 정의

// =====
// 인접 행렬(Adjacency Matrix) 기반 그래프 구조체 정의
// =====

typedef struct { // 인접 행렬 그래프를 표현하기 위한 구조체 정의 시작
    int n; // 현재 그래프에 존재하는 정점의 개수를 저장할 변수
    int adj_mat[MAX_VERTICES][MAX_VERTICES]; // 정점 사이의 간선을 0/1로 표현하는 인접 행렬
} GraphMatrix; // 이 구조체의 이름을 GraphMatrix로 사용

// 인접 행렬 그래프 초기화 함수
void init_matrix(GraphMatrix *g) { // GraphMatrix 포인터를 인자로 받아 초기화하는 함수
    g->n = 0; // 처음에는 정점 개수가 0개이므로 0으로 초기화
    for (int i = 0; i < MAX_VERTICES; i++) { // 모든 행을 순회하기 위한 반복문
        for (int j = 0; j < MAX_VERTICES; j++) { // 모든 열을 순회하기 위한 반복문
            g->adj_mat[i][j] = 0; // 모든 위치를 0으로 초기화하여 간선이 없음을 의미
        } // 내부 for문 끝
    } // 외부 for문 끝
} // init_matrix 함수 끝
```



# 인접 행렬로 구현

```
#define MAX_VERTICES 10 // 그래프에서 사용할 수 있는 최대 정점 개수를 상수로 정의

// =====
// 인접 행렬(Adjacency Matrix) 기반 그래프 구조체 정의
// =====

typedef struct { // 인접 행렬 그래프를 표현하기 위한 구조체 정의 시작
    int n; // 현재 그래프에 존재하는 정점의 개수를 저장할 변수
    int adj_mat[MAX_VERTICES][MAX_VERTICES]; // 정점 사이의 간선을 0/1로 표현하는 인접 행렬
} GraphMatrix; // 이 구조체의 이름을 GraphMatrix로 사용

// 인접 행렬 그래프 초기화 함수
void init_matrix(GraphMatrix *g) { // GraphMatrix 포인터를 인자로 받아 초기화하는 함수
    g->n = 0; // 처음에는 정점 개수가 0개이므로 0으로 초기화
    for (int i = 0; i < MAX_VERTICES; i++) { // 모든 행을 순회하기 위한 반복문
        for (int j = 0; j < MAX_VERTICES; j++) { // 모든 열을 순회하기 위한 반복문
            g->adj_mat[i][j] = 0; // 모든 위치를 0으로 초기화하여 간선이 없음을 의미
        } // 내부 for문 끝
    } // 외부 for문 끝
} // init_matrix 함수 끝
```



# 인접 행렬로 구현

```
#define MAX_VERTICES 10 // 그래프에서 사용할 수 있는 최대 정점 개수를 상수로 정의

// =====
// 인접 행렬(Adjacency Matrix) 기반 그래프 구조체 정의
// =====

typedef struct { // 인접 행렬 그래프를 표현하기 위한 구조체 정의 시작
    int n; // 현재 그래프에 존재하는 정점의 개수를 저장할 변수
    int adj_mat[MAX_VERTICES][MAX_VERTICES]; // 정점 사이의 간선을 0/1로 표현하는 인접 행렬
} GraphMatrix; // 이 구조체의 이름을 GraphMatrix로 사용

// 인접 행렬 그래프 초기화 함수
void init_matrix(GraphMatrix *g) { // GraphMatrix 포인터를 인자로 받아 초기화하는 함수
    g->n = 0; // 처음에는 정점 개수가 0개이므로 0으로 초기화
    for (int i = 0; i < MAX_VERTICES; i++) { // 모든 행을 순회하기 위한 반복문
        for (int j = 0; j < MAX_VERTICES; j++) { // 모든 열을 순회하기 위한 반복문
            g->adj_mat[i][j] = 0; // 모든 위치를 0으로 초기화하여 간선이 없음을 의미
        } // 내부 for문 끝
    } // 외부 for문 끝
} // init_matrix 함수 끝
```





# 인접 행렬로 구현

```
// 인접 행렬 그래프에 정점 하나를 추가하는 함수
void insert_vertex_matrix(GraphMatrix *g) {
    if (g->n + 1 > MAX_VERTICES) {
        printf("정점 개수 초과\n");
        return;
    }
    // 정점을 하나 추가하는 함수(정점 번호는 0부터 차례대로 부여)
    // 최대 정점 개수를 넘어가는지 검사
    // 초과할 경우 오류 메시지 출력
    // 함수 종료
    // if 문 끝
    // 정점을 하나 추가했으므로 정점 개수 증가
    // insert_vertex_matrix 함수 끝

// 인접 행렬 그래프에 간선을 추가하는 함수 (무방향 그래프 기준)
void insert_edge_matrix(GraphMatrix *g, int u, int v) {
    if (u < 0 || v < 0 || u >= g->n || v >= g->n) {
        printf("잘못된 정점 번호입니다: %d - %d\n", u, v);
        return;
    }
    // 정점 u와 v 사이에 간선을 추가하는 함수
    // 정점 번호가 유효 범위를 벗어나는지 검사
    // 범위를 벗어나면 경고 메시지 출력
    // 함수 종료
    // if 문 끝
    // u에서 v로 가는 간선이 있음을 1로 표시
    // 무방향 그래프이므로 v에서 u로도 간선이 있음을 1로 표시
    // insert_edge_matrix 함수 끝
}
```



# 인접 행렬로 구현

```
// 인접 행렬 그래프에 정점 하나를 추가하는 함수
void insert_vertex_matrix(GraphMatrix *g) {
    if (g->n + 1 > MAX_VERTICES) {
        printf("정점 개수 초과\n");
        return;
    }
    g->n++;
}

// 인접 행렬 그래프에 간선을 추가하는 함수 (무방향 그래프 기준)
void insert_edge_matrix(GraphMatrix *g, int u, int v) {
    if (u < 0 || v < 0 || u >= g->n || v >= g->n) {
        printf("잘못된 정점 번호입니다: %d - %d\n", u, v);
        return;
    }
    // 간선 추가 로직 (redacted)
}

// 정점을 하나 추가하는 함수(정점 번호는 0부터 차례대로 부여)
// 최대 정점 개수를 넘어가는지 검사
// 초과할 경우 오류 메시지 출력
// 함수 종료
// if 문 끝
// 정점을 하나 추가했으므로 정점 개수 증가
// insert_vertex_matrix 함수 끝

// 정점 u와 v 사이에 간선을 추가하는 함수
// 정점 번호가 유효 범위를 벗어나는지 검사
// 범위를 벗어나면 경고 메시지 출력
// 함수 종료
// if 문 끝
// u에서 v로 가는 간선이 있음을 1로 표시
// 무방향 그래프이므로 v에서 u로도 간선이 있음을 1로 표시
// insert_edge_matrix 함수 끝
```



# 인접 행렬로 구현

```
// 인접 행렬 그래프에 정점 하나를 추가하는 함수
void insert_vertex_matrix(GraphMatrix *g) {
    if (g->n + 1 > MAX_VERTICES) {
        printf("정점 개수 초과\n");
        return;
    }
    g->n++;
}

// 인접 행렬 그래프에 간선을 추가하는 함수 (무방향 그래프 기준)
void insert_edge_matrix(GraphMatrix *g, int u, int v) {
    if (u < 0 || v < 0 || u >= g->n || v >= g->n) {
        printf("잘못된 정점 번호입니다: %d - %d\n", u, v);
        return;
    }
    g->adj_mat[u][v] = 1;
    g->adj_mat[v][u] = 1;
}

// 정점을 하나 추가하는 함수(정점 번호는 0부터 차례대로 부여)
// 최대 정점 개수를 넘어가는지 검사
// 초과할 경우 오류 메시지 출력
// 함수 종료
// if 문 끝
// 정점을 하나 추가했으므로 정점 개수 증가
// insert_vertex_matrix 함수 끝

// 정점 u와 v 사이에 간선을 추가하는 함수
// 정점 번호가 유효 범위를 벗어나는지 검사
// 범위를 벗어나면 경고 메시지 출력
// 함수 종료
// if 문 끝
// u에서 v로 가는 간선이 있음을 1로 표시
// 무방향 그래프이므로 v에서 u로도 간선이 있음을 1로 표시
// insert_edge_matrix 함수 끝
```



# 인접 행렬로 구현

```
// 인접 행렬 그래프에 정점 하나를 추가하는 함수
void insert_vertex_matrix(GraphMatrix *g) {
    if (g->n + 1 > MAX_VERTICES) {
        printf("정점 개수 초과\n");
        return;
    }
    g->n++;
}

// 정점을 하나 추가하는 함수(정점 번호는 0부터 차례대로 부여)
// 최대 정점 개수를 넘어가는지 검사
// 초과할 경우 오류 메시지 출력
// 함수 종료
// if 문 끝
// 정점을 하나 추가했으므로 정점 개수 증가
// insert_vertex_matrix 함수 끝

// 인접 행렬 그래프에 간선을 추가하는 함수 (무방향 그래프 기준)
void insert_edge_matrix(GraphMatrix *g, int u, int v) {
    if (u < 0 || v < 0 || u >= g->n || v >= g->n) {
        printf("잘못된 정점 번호입니다: %d - %d\n", u, v);
        return;
    }
    g->adj_mat[u][v] = 1;
    g->adj_mat[v][u] = 1;
}

// 정점 u와 v 사이에 간선을 추가하는 함수
// 정점 번호가 유효 범위를 벗어나는지 검사
// 범위를 벗어나면 경고 메시지 출력
// 함수 종료
// if 문 끝
// u에서 v로 가는 간선이 있음을 1로 표시
// 무방향 그래프이므로 v에서 u로도 간선이 있음을 1로 표시
// insert_edge_matrix 함수 끝
```



# 인접 리스트로 구현 (remind)

- 연결된 정점을 단일 연결 리스트로 나열

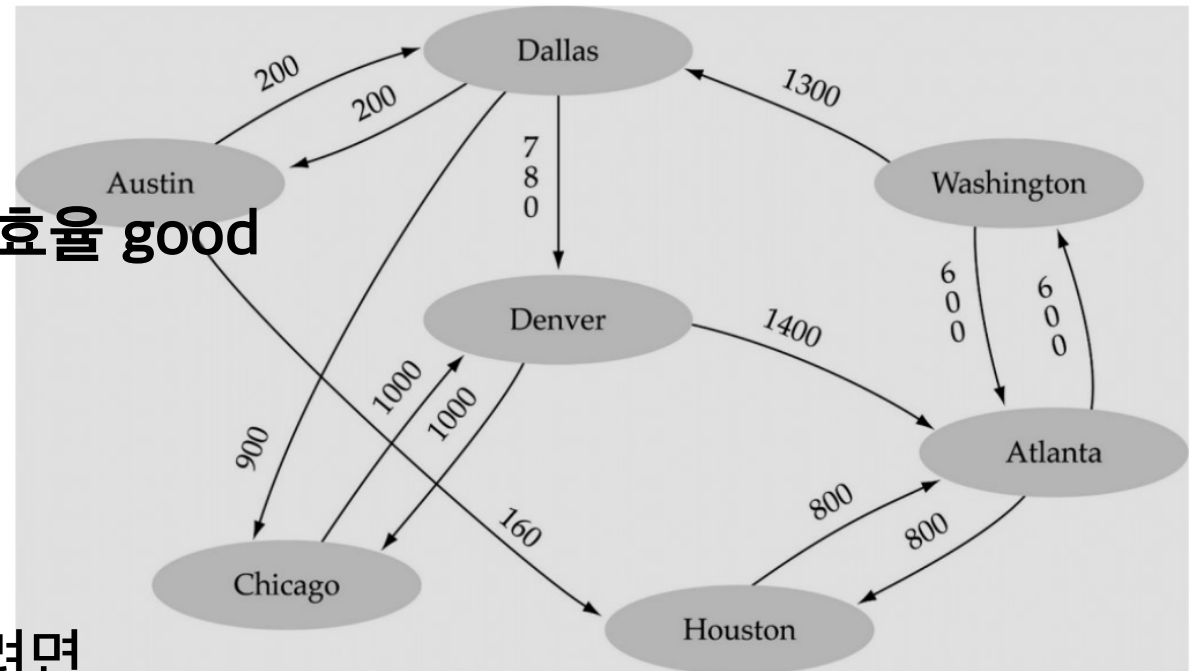
- $2 \rightarrow 0 \rightarrow 1 \rightarrow 3 \rightarrow \text{NULL}$

- 장점

- 필요한 연결만 저장하므로 메모리 효율 good
  - 간선이 적은 그래프에 적합
  - 정점의 이웃을 탐색하는 데 유리 (연결된 노드만 보면 됨)

- 단점

- 두 정점이 연결되어 있는지 확인하려면  
기준으로 잡은 정점 u에 연결된 이웃 정점의 수 만큼 탐색 필요
  - 구현이 행렬보다 복잡함







# 인접 리스트로 구현

```
// =====  
// 인접 리스트(Adjacency List) 기반 그래프 구조체 정의  
// =====  
  
// 인접 리스트에서 사용할 노드 구조체 정의  
typedef struct GraphNode {  
    int vertex;  
    struct GraphNode *link;  
} GraphNode;  
  
// 인접 리스트 그래프 구조체  
typedef struct {  
    int n;  
    GraphNode *adj_list[MAX_VERTICES];  
} GraphList;  
  
// 인접 리스트의 한 노드를 표현하는 구조체 정의 시작  
// 이 노드가 나타내는 이웃 정점 번호  
// 다음 이웃 노드를 가리키는 포인터 (단순 연결 리스트 구조)  
// 이 구조체의 이름을 GraphNode로 사용  
  
// 인접 리스트 기반 그래프 구조체 정의 시작  
// 현재 그래프에 존재하는 정점의 개수  
// 각 정점에 대한 인접 리스트의 시작 포인터 배열  
// 이 구조체의 이름을 GraphList로 사용
```



# 인접 리스트로 구현

```
// 인접 리스트에서 사용할 노드 구조체 정의
typedef struct GraphNode {
    int vertex;
    struct GraphNode *link;
} GraphNode;

// 인접 리스트 그래프 구조체
typedef struct {
    int n;
    GraphNode *adj_list[MAX_VERTICES];
} GraphList;
```

```
// 인접 리스트의 한 노드를 표현하는 구조체 정의 시작
// 이 노드가 나타내는 이웃 정점 번호
// 다음 이웃 노드를 가리키는 포인터 (단순 연결 리스트 구조)
// 이 구조체의 이름을 GraphNode로 사용
```

```
// 인접 리스트 기반 그래프 구조체 정의 시작
// 현재 그래프에 존재하는 정점의 개수
// 각 정점에 대한 인접 리스트의 시작 포인터 배열
// 이 구조체의 이름을 GraphList로 사용
```

```
// 인접 리스트 그래프 초기화 함수
void init_list(GraphList *g) {
    g->n = 0;
    for (int i = 0; i < MAX_VERTICES; i++) {
        g->adj_list[i] = NULL;
    }
}
```

```
// GraphList 포인터를 받아 그래프를 초기화하는 함수
// 처음에는 정점 개수가 0개이므로 0으로 초기화
// 모든 정점 인덱스를 순회하는 반복문
// 각 정점의 인접 리스트 시작 포인터를 NULL로 초기화
// for문 끝
// init_list 함수 끝
```



# 인접 리스트로 구현

// 인접 리스트에서 사용할 노드 구조체 정의

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

// 인접 리스트 그래프 구조체

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

// 인접 리스트의 한 노드를 표현하는 구조체 정의 시작

// 이 노드가 나타내는 이웃 정점 번호

// 다음 이웃 노드를 가리키는 포인터 (단순 연결 리스트 구조)

// 이 구조체의 이름을 GraphNode로 사용

// 인접 리스트 기반 그래프 구조체 정의 시작

// 현재 그래프에 존재하는 정점의 개수

// 각 정점에 대한 인접 리스트의 시작 포인터 배열

// 이 구조체의 이름을 GraphList로 사용

// 인접 리스트 그래프 초기화 함수

```
void init_list(GraphList *g) {  
    g->n = 0;  
    for (int i = 0; i < [REDACTED]; i++) {  
        [REDACTED] = NULL;  
    }  
}
```

// GraphList 포인터를 받아 그래프를 초기화하는 함수

// 처음에는 정점 개수가 0개이므로 0으로 초기화

// 모든 정점 인덱스를 순회하는 반복문

// 각 정점의 인접 리스트 시작 포인터를 NULL로 초기화

// for문 끝

// init\_list 함수 끝





# 인접 리스트로 구현

```
// 인접 리스트에서 사용할 노드 구조체 정의
typedef struct GraphNode {
    int vertex;
    struct GraphNode *link;
} GraphNode;

// 인접 리스트 그래프 구조체
typedef struct {
    int n;
    GraphNode *adj_list[MAX_VERTICES];
} GraphList;
```

```
// 인접 리스트의 한 노드를 표현하는 구조체 정의 시작
// 이 노드가 나타내는 이웃 정점 번호
// 다음 이웃 노드를 가리키는 포인터 (단순 연결 리스트 구조)
// 이 구조체의 이름을 GraphNode로 사용
```

```
// 인접 리스트 기반 그래프 구조체 정의 시작
// 현재 그래프에 존재하는 정점의 개수
// 각 정점에 대한 인접 리스트의 시작 포인터 배열
// 이 구조체의 이름을 GraphList로 사용
```

```
// 인접 리스트 그래프 초기화 함수
void init_list(GraphList *g) {
    g->n = 0;
    for (int i = 0; i < MAX_VERTICES; i++) {
        [REDACTED] = NULL;
    }
}
```

```
// GraphList 포인터를 받아 그래프를 초기화하는 함수
// 처음에는 정점 개수가 0개이므로 0으로 초기화
// 모든 정점 인덱스를 순회하는 반복문
// 각 정점의 인접 리스트 시작 포인터를 NULL로 초기화
// for문 끝
// init_list 함수 끝
```



# 인접 리스트로 구현

// 인접 리스트에서 사용할 노드 구조체 정의

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

// 인접 리스트 그래프 구조체

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

// 인접 리스트의 한 노드를 표현하는 구조체 정의 시작

// 이 노드가 나타내는 이웃 정점 번호

// 다음 이웃 노드를 가리키는 포인터 (단순 연결 리스트 구조)

// 이 구조체의 이름을 GraphNode로 사용

// 인접 리스트 기반 그래프 구조체 정의 시작

// 현재 그래프에 존재하는 정점의 개수

// 각 정점에 대한 인접 리스트의 시작 포인터 배열

// 이 구조체의 이름을 GraphList로 사용

// 인접 리스트 그래프 초기화 함수

```
void init_list(GraphList *g) {  
    g->n = 0;  
    for (int i = 0; i < MAX_VERTICES; i++) {  
        g->adj_list[i] = NULL;  
    }  
}
```

// GraphList 포인터를 받아 그래프를 초기화하는 함수

// 처음에는 정점 개수가 0개이므로 0으로 초기화

// 모든 정점 인덱스를 순회하는 반복문

// 각 정점의 인접 리스트 시작 포인터를 NULL로 초기화

// for문 끝

// init\_list 함수 끝



# 인접 리스트로 구현

```
// 인접 리스트 그래프에 정점 하나를 추가하는 함수
void insert_vertex_list(GraphList *g) {
    if ( [REDACTED] > MAX_VERTICES) {
        printf("정점 개수 초과\n");
        return;
    }
    g->adj_list[g->n] = NULL;
    [REDACTED]
}
```

```
// 인접 리스트 그래프에 새 정점을 추가하는 함수
// 정점 개수가 최대치를 넘는지 검사
// 초과 시 오류 메시지 출력
// 함수 종료
// if 문 끝
// 새로 추가되는 정점의 인접 리스트를 NULL로 초기화
// 정점 개수를 1 증가
// insert_vertex_list 함수 끝
```



# 인접 리스트로 구현

```
// 인접 리스트 그래프에 정점 하나를 추가하는 함수
void insert_vertex_list(GraphList *g) {
    if (g->n + 1 > MAX_VERTICES) {
        printf("정점 개수 초과\n");
        return;
    }
    [redacted] = NULL;
    [redacted]
}

// 인접 리스트 그래프에 새 정점을 추가하는 함수
// 정점 개수가 최대치를 넘는지 검사
// 초과 시 오류 메시지 출력
// 함수 종료
// if 문 끝
// 새로 추가되는 정점의 인접 리스트를 NULL로 초기화
// 정점 개수를 1 증가
// insert_vertex_list 함수 끝
```



# 인접 리스트로 구현

```
// 인접 리스트 그래프에 정점 하나를 추가하는 함수
void insert_vertex_list(GraphList *g) {
    if (g->n + 1 > MAX_VERTICES) {
        printf("정점 개수 초과\n");
        return;
    }
    g->adj_list[g->n] = NULL;
    [REDACTED]
}

// 인접 리스트 그래프에 새 정점을 추가하는 함수
// 정점 개수가 최대치를 넘는지 검사
// 초과 시 오류 메시지 출력
// 함수 종료
// if 문 끝
// 새로 추가되는 정점의 인접 리스트를 NULL로 초기화
// 정점 개수를 1 증가
// insert_vertex_list 함수 끝
```



# 인접 리스트로 구현

```
// 인접 리스트 그래프에 정점 하나를 추가하는 함수
void insert_vertex_list(GraphList *g) {
    if (g->n + 1 > MAX_VERTICES) {
        printf("정점 개수 초과\n");
        return;
    }
    g->adj_list[g->n] = NULL;
    g->n++;
}
```

```
// 인접 리스트 그래프에 새 정점을 추가하는 함수
// 정점 개수가 최대치를 넘는지 검사
// 초과 시 오류 메시지 출력
// 함수 종료
// if 문 끝
// 새로 추가되는 정점의 인접 리스트를 NULL로 초기화
// 정점 개수를 1 증가
// insert_vertex_list 함수 끝
```



# 인접 리스트로 구현

```
// 인접 리스트 그래프에 간선을 추가하는 함수 (무방향 그래프 기준)
void insert_edge_list(GraphList *g, int u, int v) { // 정점 u와 v 사이에 간선을 추가하는 함수
    if (u < 0 || v < 0 || u >= [REDACTED] || v >= [REDACTED]) { // 정점 번호가 유효한지 검사
        printf("잘못된 정점 번호입니다: %d - %d\n", u, v); // 잘못된 경우 경고 메시지 출력
        return; // 함수 종료
    } // if 문 끝

    // u -> v 방향 간선을 인접 리스트에 삽입
    GraphNode *node1 = (GraphNode *)malloc(sizeof(GraphNode)); // u의 인접 리스트에 들어갈 새 노드를 동적 할당
    node1->vertex = v; // 새 노드가 가리키는 이웃 정점 번호를 v로 설정
    node1->link = [REDACTED] // 새 노드를 기존 리스트의 맨 앞에 연결
    g->adj_list[u] = [REDACTED] // u의 인접 리스트 시작 포인터를 새 노드로 갱신

    // v -> u 방향 간선을 인접 리스트에 삽입 (무방향 그래프이므로 반대 방향도 추가)
    GraphNode *node2 = (GraphNode *)malloc(sizeof(GraphNode)); // v의 인접 리스트에 들어갈 새 노드 동적 할당
    [REDACTED] // 새 노드가 가리키는 이웃 정점 번호를 u로 설정
    [REDACTED] // 새 노드를 기존 리스트의 맨 앞에 연결
    [REDACTED] // v의 인접 리스트 시작 포인터를 새 노드로 갱신
    // insert_edge_list 함수 끝
}
```





# 인접 리스트로 구현

```
// 인접 리스트 그래프에 간선을 추가하는 함수 (무방향 그래프 기준)
void insert_edge_list(GraphList *g, int u, int v) { // 정점 u와 v 사이에 간선을 추가하는 함수
    if (u < 0 || v < 0 || u >= g->n || v >= g->n) { // 정점 번호가 유효한지 검사
        printf("잘못된 정점 번호입니다: %d - %d\n", u, v); // 잘못된 경우 경고 메시지 출력
        return; // 함수 종료
    } // if 문 끝

    // u -> v 방향 간선을 인접 리스트에 삽입
    GraphNode *node1 = (GraphNode *)malloc(sizeof(GraphNode)); // u의 인접 리스트에 들어갈 새 노드를 동적 할당
    node1->vertex = v; // 새 노드가 가리키는 이웃 정점 번호를 v로 설정
    node1->link = [redacted] // 새 노드를 기존 리스트의 맨 앞에 연결
    g->adj_list[u] = [redacted] // u의 인접 리스트 시작 포인터를 새 노드로 갱신

    // v -> u 방향 간선을 인접 리스트에 삽입 (무방향 그래프이므로 반대 방향도 추가)
    GraphNode *node2 = (GraphNode *)malloc(sizeof(GraphNode)); // v의 인접 리스트에 들어갈 새 노드 동적 할당
    [redacted] // 새 노드가 가리키는 이웃 정점 번호를 u로 설정
    [redacted] // 새 노드를 기존 리스트의 맨 앞에 연결
    [redacted] // v의 인접 리스트 시작 포인터를 새 노드로 갱신
    // insert_edge_list 함수 끝
}
```





# 인접 리스트로 구현

```
// 인접 리스트 그래프에 간선을 추가하는 함수 (무방향 그래프 기준)
void insert_edge_list(GraphList *g, int u, int v) { // 정점 u와 v 사이에 간선을 추가하는 함수
    if (u < 0 || v < 0 || u >= g->n || v >= g->n) { // 정점 번호가 유효한지 검사
        printf("잘못된 정점 번호입니다: %d - %d\n", u, v); // 잘못된 경우 경고 메시지 출력
        return; // 함수 종료
    } // if 문 끝

    // u -> v 방향 간선을 인접 리스트에 삽입
    GraphNode *node1 = (GraphNode *)malloc(sizeof(GraphNode)); // u의 인접 리스트에 들어갈 새 노드를 동적 할당
    node1->vertex = v; // 새 노드가 가리키는 이웃 정점 번호를 v로 설정
    node1->link = g->adj_list[u]; // 새 노드를 기존 리스트의 맨 앞에 연결
    g->adj_list[u] = node1; // u의 인접 리스트 시작 포인터를 새 노드로 갱신

    // v -> u 방향 간선을 인접 리스트에 삽입 (무방향 그래프이므로 반대 방향도 추가)
    GraphNode *node2 = (GraphNode *)malloc(sizeof(GraphNode)); // v의 인접 리스트에 들어갈 새 노드 동적 할당
    node2->vertex = u; // 새 노드가 가리키는 이웃 정점 번호를 u로 설정
    node2->link = g->adj_list[v]; // 새 노드를 기존 리스트의 맨 앞에 연결
    g->adj_list[v] = node2; // v의 인접 리스트 시작 포인터를 새 노드로 갱신
} // insert_edge_list 함수 끝
```



# 인접 리스트로 구현

```
// 인접 리스트 그래프에 간선을 추가하는 함수 (무방향 그래프 기준)
void insert_edge_list(GraphList *g, int u, int v) { // 정점 u와 v 사이에 간선을 추가하는 함수
    if (u < 0 || v < 0 || u >= g->n || v >= g->n) { // 정점 번호가 유효한지 검사
        printf("잘못된 정점 번호입니다: %d - %d\n", u, v); // 잘못된 경우 경고 메시지 출력
        return; // 함수 종료
    } // if 문 끝

    // u -> v 방향 간선을 인접 리스트에 삽입
    GraphNode *node1 = (GraphNode *)malloc(sizeof(GraphNode)); // u의 인접 리스트에 들어갈 새 노드를 동적 할당
    node1->vertex = v; // 새 노드가 가리키는 이웃 정점 번호를 v로 설정
    node1->link = g->adj_list[u]; // 새 노드를 기존 리스트의 맨 앞에 연결
    g->adj_list[u] = node1; // u의 인접 리스트 시작 포인터를 새 노드로 갱신

    // v -> u 방향 간선을 인접 리스트에 삽입 (무방향 그래프이므로 반대 방향도 추가)
    GraphNode *node2 = (GraphNode *)malloc(sizeof(GraphNode)); // v의 인접 리스트에 들어갈 새 노드 동적 할당
    node2->vertex = u; // 새 노드가 가리키는 이웃 정점 번호를 u로 설정
    node2->link = g->adj_list[v]; // 새 노드를 기존 리스트의 맨 앞에 연결
    g->adj_list[v] = node2; // v의 인접 리스트 시작 포인터를 새 노드로 갱신
} // insert_edge_list 함수 끝
```



# 인접 리스트로 구현

```
// 인접 리스트 그래프에 간선을 추가하는 함수 (무방향 그래프 기준)
void insert_edge_list(GraphList *g, int u, int v) { // 정점 u와 v 사이에 간선을 추가하는 함수
    if (u < 0 || v < 0 || u >= g->n || v >= g->n) { // 정점 번호가 유효한지 검사
        printf("잘못된 정점 번호입니다: %d - %d\n", u, v); // 잘못된 경우 경고 메시지 출력
        return; // 함수 종료
    } // if 문 끝

    // u -> v 방향 간선을 인접 리스트에 삽입
    GraphNode *node1 = (GraphNode *)malloc(sizeof(GraphNode)); // u의 인접 리스트에 들어갈 새 노드를 동적 할당
    node1->vertex = v; // 새 노드가 가리키는 이웃 정점 번호를 v로 설정
    node1->link = g->adj_list[u]; // 새 노드를 기존 리스트의 맨 앞에 연결
    g->adj_list[u] = node1; // u의 인접 리스트 시작 포인터를 새 노드로 갱신

    // v -> u 방향 간선을 인접 리스트에 삽입 (무방향 그래프이므로 반대 방향도 추가)
    GraphNode *node2 = (GraphNode *)malloc(sizeof(GraphNode)); // v의 인접 리스트에 들어갈 새 노드 동적 할당
    node2->vertex = u; // 새 노드가 가리키는 이웃 정점 번호를 u로 설정
    node2->link = g->adj_list[v]; // 새 노드를 기존 리스트의 맨 앞에 연결
    g->adj_list[v] = node2; // v의 인접 리스트 시작 포인터를 새 노드로 갱신
} // insert_edge_list 함수 끝
```



# 인접 리스트로 구현

```
// 인접 리스트 그래프에 간선을 추가하는 함수 (무방향 그래프 기준)
void insert_edge_list(GraphList *g, int u, int v) { // 정점 u와 v 사이에 간선을 추가하는 함수
    if (u < 0 || v < 0 || u >= g->n || v >= g->n) { // 정점 번호가 유효한지 검사
        printf("잘못된 정점 번호입니다: %d - %d\n", u, v); // 잘못된 경우 경고 메시지 출력
        return; // 함수 종료
    } // if 문 끝

    // u -> v 방향 간선을 인접 리스트에 삽입
    GraphNode *node1 = (GraphNode *)malloc(sizeof(GraphNode)); // u의 인접 리스트에 들어갈 새 노드를 동적 할당
    node1->vertex = v; // 새 노드가 가리키는 이웃 정점 번호를 v로 설정
    node1->link = g->adj_list[u]; // 새 노드를 기존 리스트의 맨 앞에 연결
    g->adj_list[u] = node1; // u의 인접 리스트 시작 포인터를 새 노드로 갱신

    // v -> u 방향 간선을 인접 리스트에 삽입 (무방향 그래프이므로 반대 방향도 추가)
    GraphNode *node2 = (GraphNode *)malloc(sizeof(GraphNode)); // v의 인접 리스트에 들어갈 새 노드 동적 할당
    node2->vertex = u; // 새 노드가 가리키는 이웃 정점 번호를 u로 설정
    node2->link = g->adj_list[v]; // 새 노드를 기존 리스트의 맨 앞에 연결
    g->adj_list[v] = node2; // v의 인접 리스트 시작 포인터를 새 노드로 갱신
} // insert_edge_list 함수 끝
```





# 인접 리스트로 구현

```
// 인접 리스트 그래프에 간선을 추가하는 함수 (무방향 그래프 기준)
void insert_edge_list(GraphList *g, int u, int v) { // 정점 u와 v 사이에 간선을 추가하는 함수
    if (u < 0 || v < 0 || u >= g->n || v >= g->n) { // 정점 번호가 유효한지 검사
        printf("잘못된 정점 번호입니다: %d - %d\n", u, v); // 잘못된 경우 경고 메시지 출력
        return; // 함수 종료
    } // if 문 끝

    // u -> v 방향 간선을 인접 리스트에 삽입
    GraphNode *node1 = (GraphNode *)malloc(sizeof(GraphNode)); // u의 인접 리스트에 들어갈 새 노드를 동적 할당
    node1->vertex = v; // 새 노드가 가리키는 이웃 정점 번호를 v로 설정
    node1->link = g->adj_list[u]; // 새 노드를 기존 리스트의 맨 앞에 연결
    g->adj_list[u] = node1; // u의 인접 리스트 시작 포인터를 새 노드로 갱신

    // v -> u 방향 간선을 인접 리스트에 삽입 (무방향 그래프이므로 반대 방향도 추가)
    GraphNode *node2 = (GraphNode *)malloc(sizeof(GraphNode)); // v의 인접 리스트에 들어갈 새 노드 동적 할당
    node2->vertex = u; // 새 노드가 가리키는 이웃 정점 번호를 u로 설정
    node2->link = g->adj_list[v]; // 새 노드를 기존 리스트의 맨 앞에 연결
    g->adj_list[v] = node2; // v의 인접 리스트 시작 포인터를 새 노드로 갱신
} // insert_edge_list 함수 끝
```



# Main.c 실행

```
// 정점 4개(0, 1, 2, 3)를 추가하는 예제
for (int i = 0; i < 4; i++) {
    insert_vertex_matrix(&g_mat);
    insert_vertex_list(&g_list);
}

// 간선들을 추가하여 간단한 무방향 그래프를 구성
// 예: 0-1, 0-2, 1-2, 2-3
insert_edge_matrix(&g_mat, 0, 1);
insert_edge_matrix(&g_mat, 0, 2);
insert_edge_matrix(&g_mat, 1, 2);
insert_edge_matrix(&g_mat, 2, 3);

insert_edge_list(&g_list, 0, 1);
insert_edge_list(&g_list, 0, 2);
insert_edge_list(&g_list, 1, 2);
insert_edge_list(&g_list, 2, 3);
```

```
=== Adjacency Matrix (인접 행렬) ===
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0
```

```
=== Adjacency List (인접 리스트) ===
정점 0: 2 -> 1 -> NULL
정점 1: 2 -> 0 -> NULL
정점 2: 3 -> 1 -> 0 -> NULL
정점 3: 2 -> NULL
```

값들을 직접 바꿔가며 실행해 보세요



# 수고하셨습니다

자료구조 실습 12/02

EOF