**로고, 폰트, 텍스트, 상징이(가) 표시된 사진

자동 생성된 설명**

**[데이터구조2 리포트]**

**이름: 황지상 학번: 20224060**

**과제#3 방향그래프**

1. **코드**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define MAX\_VERTICES 50 // 최대 정점 수를 정의

#define MAX 50 // 배열의 최대 크기를 정의

#define TRUE 1 // 참을 나타내는 상수 정의

#define FALSE 0 // 거짓을 나타내는 상수 정의

// 그래프의 각 정점을 나타내는 구조체

typedef struct GraphNode {

int vertex; // 정점 번호를 저장

struct GraphNode\* link; // 다음 정점을 연결하기 위한 포인터

} GraphNode;

// 인접 행렬을 사용한 그래프 구조체

typedef struct GraphType {

int n; // 정점의 개수를 저장

int adj\_mat[MAX\_VERTICES][MAX\_VERTICES]; // 정점 간 연결 여부를 저장하는 인접 행렬

} GraphType;

// 그래프 초기화 함수

void init(GraphType\* g) {

int r, c;

g->n = 0; // 정점의 개수 초기화

for (r = 0; r < MAX\_VERTICES; r++) {

for (c = 0; c < MAX\_VERTICES; c++) {

g->adj\_mat[r][c] = 0; // 인접 행렬 초기화

}

}

}

// 정점을 그래프에 추가하는 함수

void insert\_vertex(GraphType\* g, int v) {

if (((g->n) + 1) > MAX\_VERTICES) {

fprintf(stderr, "그래프: 정점의 개수 초과"); // 정점 개수 초과 시 에러 출력

return;

}

g->n++;

}

// 간선을 그래프에 추가하는 함수

void insert\_edge(GraphType\* g, int start, int end) {

if (start >= g->n || end >= g->n) {

fprintf(stderr, "그래프: 정점의 번호 오류"); // 정점 번호 오류 시 에러 출력

return;

}

g->adj\_mat[start][end] = 1; // 인접 행렬에서 해당 간선을 활성화

}

// 인접 행렬을 출력하는 함수

void print\_adj\_mat(GraphType\* g) {

for (int i = 0; i < g->n; i++) {

printf("%c\t", 'A' + i); // 정점 이름을 출력

for (int j = 0; j < g->n; j++) {

printf("%2d ", g->adj\_mat[i][j]); // 인접 행렬의 값을 출력

}

printf("\n");

}

}

int visited[MAX\_VERTICES]; // 방문 여부를 저장하는 배열

// 인접 행렬을 사용한 깊이 우선 탐색(DFS) 함수

void dfs\_mat(GraphType\* g, int v) {

int w;

visited[v] = TRUE; // 정점을 방문한 것으로 표시

printf("%c ", 'A' + v); // 방문한 정점을 출력

for (w = 0; w < g->n; w++) {

if (g->adj\_mat[v][w] && !visited[w]) {

dfs\_mat(g, w); // 방문하지 않은 인접 정점을 재귀적으로 탐색

}

}

}

// 인접 리스트를 사용한 그래프 구조체

typedef struct GraphType1 {

int n; // 정점의 개수를 저장

GraphNode\* adj\_list[MAX]; // 각 정점의 인접 리스트를 저장하는 배열

} GraphType1;

// 그래프 초기화 함수

void init1(GraphType1\* g) {

int v;

g->n = 0; // 정점의 개수 초기화

for (v = 0; v < MAX; v++) {

g->adj\_list[v] = NULL; // 인접 리스트 초기화

}

}

// 정점을 인접 리스트에 추가하는 함수

void insert\_list\_vertex(GraphType1\* g, int v) {

if (((g->n) + 1) > MAX) {

fprintf(stderr, "그래프: 정점의 개수 초과"); // 정점 개수 초과 시 에러 출력

return;

}

g->n++;

}

// 간선을 인접 리스트에 추가하는 함수

void insert\_list\_edge(GraphType1\* g, int u, int v) {

GraphNode\* node;

if (u >= g->n || v >= g->n) {

fprintf(stderr, "그래프: 정점 번호 오류"); // 정점 번호 오류 시 에러 출력

return;

}

node = (GraphNode\*)malloc(sizeof(GraphNode)); // 새로운 노드 생성

node->vertex = v; // 노드에 정점 번호 저장

node->link = g->adj\_list[u]; // 노드를 인접 리스트에 연결

g->adj\_list[u] = node;

}

// 인접 리스트를 출력하는 함수

void print\_adj\_list(GraphType1\* g) {

for (int i = 0; i < g->n; i++) {

GraphNode\* p = g->adj\_list[i];

printf("정점 %c의 인접 리스트", 'A' + i); // 정점 이름과 인접 리스트 시작을 출력

while (p != NULL) {

printf("-> %c", 'A' + p->vertex); // 인접 정점을 출력

p = p->link; // 다음 인접 정점으로 이동

}

printf("\n");

}

}

int visited1[MAX\_VERTICES]; // 방문 여부를 저장하는 배열

// 인접 리스트를 사용한 깊이 우선 탐색(DFS) 함수

void dfs\_list(GraphType1\* g, int v) {

GraphNode\* w;

visited1[v] = TRUE; // 정점을 방문한 것으로 표시

printf("정점 %c -> ", 'A' + v); // 방문한 정점을 출력

for (w = g->adj\_list[v]; w;w = w->link) {

if (!visited1[w->vertex])

dfs\_list(g, w->vertex); // 방문하지 않은 인접 정점을 재귀적으로 탐색

}

}

int main(void) {

GraphType\* g; // 인접 행렬을 사용하는 그래프

GraphType1\* gg; // 인접 리스트를 사용하는 그래프

g = (GraphType\*)malloc(sizeof(GraphType)); // 인접 행렬 그래프 메모리 할당

gg = (GraphType1\*)malloc(sizeof(GraphType1)); // 인접 리스트 그래프 메모리 할당

init(g); // 그래프 초기화

init1(gg); // 그래프 초기화

// 7개의 정점을 그래프에 추가

for (int i = 0; i < 7; i++) {

insert\_vertex(g, i);

}

// 간선 정보를 추가하여 그래프 연결

insert\_edge(g, 0, 1);

insert\_edge(g, 0, 2);

insert\_edge(g, 0, 3);

insert\_edge(g, 0, 4);

insert\_edge(g, 0, 5);

insert\_edge(g, 0, 6);

insert\_edge(g, 1, 0);

insert\_edge(g, 1, 6);

insert\_edge(g, 2, 0);

insert\_edge(g, 2, 1);

insert\_edge(g, 2, 2);

insert\_edge(g, 2, 4);

insert\_edge(g, 2, 5);

insert\_edge(g, 2, 6);

insert\_edge(g, 3, 2);

insert\_edge(g, 3, 4);

insert\_edge(g, 4, 2);

insert\_edge(g, 4, 3);

insert\_edge(g, 4, 4);

insert\_edge(g, 4, 5);

insert\_edge(g, 5, 2);

insert\_edge(g, 5, 5);

insert\_edge(g, 5, 6);

insert\_edge(g, 6, 1);

insert\_edge(g, 6, 2);

insert\_edge(g, 6, 3);

insert\_edge(g, 6, 4);

insert\_edge(g, 6, 5);

printf("\n");

print\_adj\_mat(g); // 인접 행렬 출력

printf("인접 행렬 DFS: ");

// 방문 배열 초기화 및 DFS 실행

for (int i = 0; i < 7; i++) {

visited[i] = FALSE; // visited1 배열 초기화

}

for (int i = 0; i < 7; i++) {

if (!visited[i]) {

dfs\_mat(g, i);

}

}

printf("\n \n");

// 7개의 정점을 그래프에 추가

for (int i = 0; i < g->n; i++) {

insert\_list\_vertex(gg, i);

}

// 인접 행렬을 기반으로 인접 리스트 정보 추가

for (int i = 0; i < g->n;i++) {

for (int j = 0; j < g->n; j++) {

if (g->adj\_mat[i][j] == 1) {

insert\_list\_edge(gg, i, j);

}

}

}

print\_adj\_list(gg); // 인접 리스트 출력

printf("인접 리스트 DFS: ");

// 방문 배열 초기화 및 DFS 실행

for (int i = 0; i < 7; i++) {

visited1[i] = FALSE; // visited2 배열 초기화

}

for (int i = 0; i < 7; i++) {

if (!visited1[i]) {

dfs\_list(gg, i);

}

}

free(g); // 그래프 메모리 해제

free(gg); // 그래프 메모리 해제

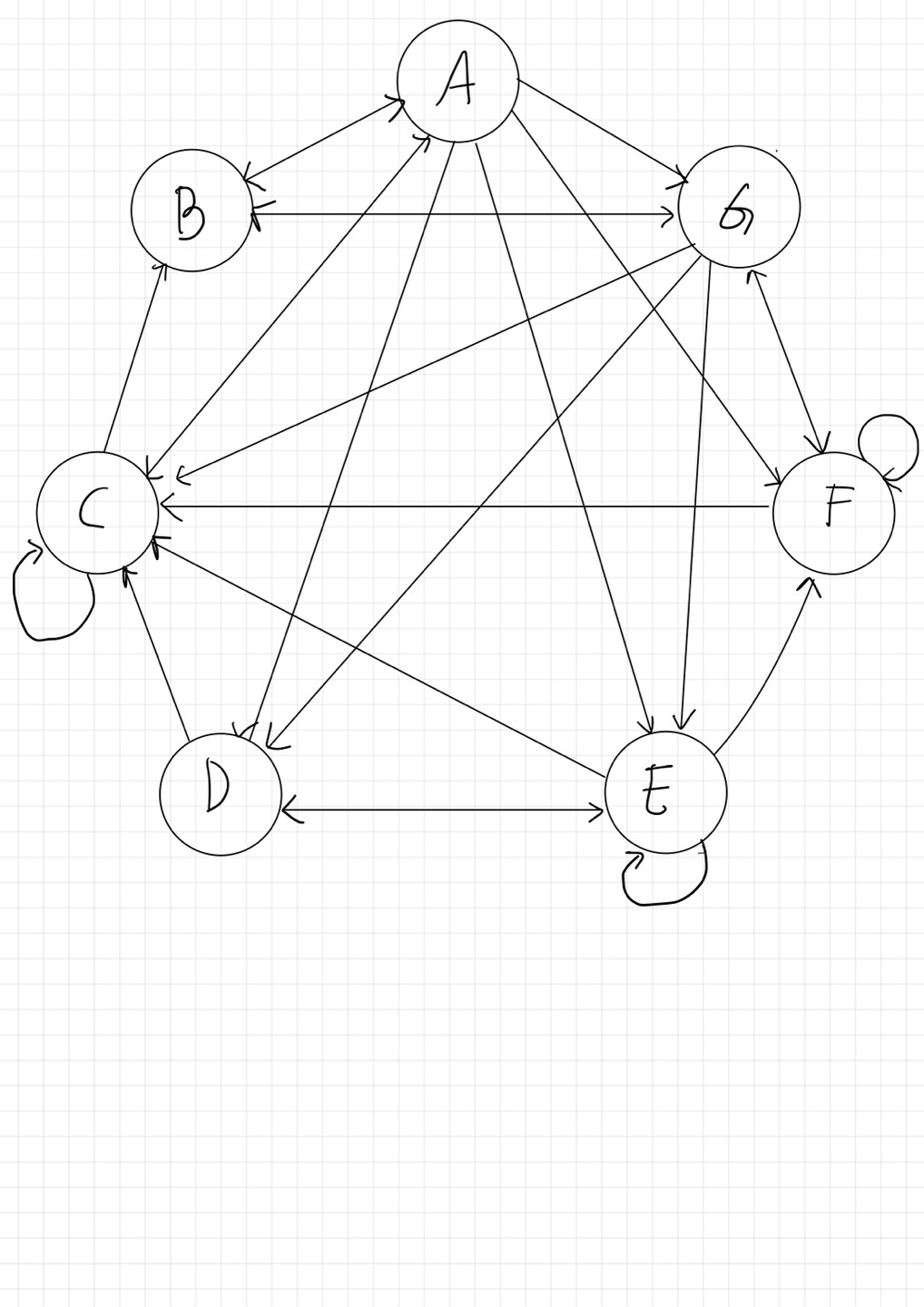
return 0;

}

1. **실행 결과**

**텍스트, 스크린샷, 블랙, 흑백이(가) 표시된 사진

자동 생성된 설명**

****