**로고, 폰트, 텍스트, 상징이(가) 표시된 사진

자동 생성된 설명**

**[데이터구조2 리포트]**

**이름: 황지상 학번: 20224060**

**과제#3\_Graph\_List\_DFS**

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

g->adj\_mat[end][start] = 1; // 무방향 그래프이므로 양방향으로 설정

}

// 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");

}

}

// 방문 여부를 저장하는 배열

int visited[MAX\_VERTICES];

// 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

}

}

}

// 그래프의 인접 리스트를 표현하기 위한 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; // 그래프의 정점 개수 초기화

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

g->adj\_list[v] = NULL; // 각 정점의 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++; // 정점 개수 증가

}

// 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; // 정점 u의 인접 리스트의 시작을 새로운 노드로 갱신

}

// 인접 리스트를 출력하는 함수

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

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

GraphNode\* p = g->adj\_list[i]; // 현재 정점의 인접 리스트 시작을 가리키는 포인터

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

while (p != NULL) {

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

p = p->link; // 다음 인접 정점으로 이동

}

printf("\n");

}

}

// 방문 여부를 저장하는 배열

int visited1[MAX\_VERTICES];

// 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

}

}

// 두 정수 x와 y가 같은지를 확인하는 함수

int random\_same(int x, int y) {

if (x == y) {

return 0; // x와 y가 같으면 0을 반환

}

return 1; // x와 y가 다르면 1을 반환

}

// 그래프에 무작위 간선을 추가하는 함수

void adj\_produce(GraphType\* g, int num1, int line) {

srand(time(NULL)); // 난수 발생을 위한 시드 설정

int\* line\_list\_x = (int\*)malloc(sizeof(int) \* line); // 무작위 간선의 시작 정점을 저장하는 배열

int\* line\_list\_y = (int\*)malloc(sizeof(int) \* line); // 무작위 간선의 끝 정점을 저장하는 배열

int index = 0; // 배열 인덱스 변수 초기화

for (int i = 0; i < line; i++) { // 주어진 간선 개수만큼 반복

int randomx = rand() % num1; // 0부터 (num1-1) 사이의 난수 생성 (시작 정점)

int randomy = rand() % num1; // 0부터 (num1-1) 사이의 난수 생성 (끝 정점)

int chk = 0; // 간선 중복 확인을 위한 변수 초기화

if (random\_same(randomx, randomy)) { // 시작 정점과 끝 정점이 다를 경우

for (int j = 0; j < index; j++) { // 이미 생성된 간선과 비교

if ((randomx == line\_list\_x[j] && randomy == line\_list\_y[j]) ||

(randomx == line\_list\_y[j] && randomy == line\_list\_x[j]) || (randomx == randomy)) {

chk = 1; // 중복된 간선이 있으면 chk를 1로 설정

i--; // 중복된 경우 현재 반복을 다시 수행하도록 설정

break; // 반복 종료

}

}

if (chk != 1) { // 중복된 간선이 없는 경우

if (randomx < randomy) {

line\_list\_x[index] = randomx; // 작은 정점을 시작 정점으로 저장

line\_list\_y[index] = randomy; // 큰 정점을 끝 정점으로 저장

}

else {

line\_list\_x[index] = randomy; // 큰 정점을 시작 정점으로 저장

line\_list\_y[index] = randomx; // 작은 정점을 끝 정점으로 저장

}

index++; // 배열 인덱스 증가

insert\_edge(g, randomx, randomy); // 그래프에 간선 추가

}

}

else {

i--; // 시작 정점과 끝 정점이 같은 경우 현재 반복을 다시 수행하도록 설정

}

}

free(line\_list\_x); // 동적으로 할당한 배열 메모리 해제

free(line\_list\_y); // 동적으로 할당한 배열 메모리 해제

}

int main(void) {

int num1 = 0, line = 0;

GraphType\* g; // 인접 행렬을 사용할 그래프 포인터 선언

GraphType1\* gg; // 인접 리스트를 사용할 그래프 포인터 선언

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

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

init(g); // 인접 행렬 그래프 초기화 함수 호출

init1(gg); // 인접 리스트 그래프 초기화 함수 호출

printf("정점의 개수는?");

scanf("%d", &num1); // 사용자로부터 정점의 개수 입력 받음

for (int i = 0; i < num1; i++) {

insert\_vertex(g, i); // 인접 행렬 그래프에 정점 추가

}

printf("랜덤 연결 그래프 생성\n");

printf("간선 수:");

scanf("%d", &line); // 사용자로부터 간선의 개수 입력 받음

adj\_produce(g, num1, line); // 인접 행렬 그래프에 무작위 간선 추가

print\_adj\_mat(g); // 인접 행렬 그래프 출력

printf("인접 행렬 DFS: ");

for (int i = 0; i < num1; i++) {

if (!visited[i]) {

dfs\_mat(g, i); // 인접 행렬 그래프에 대한 DFS 수행

}

}

printf("\n \n");

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

insert\_list\_vertex(gg, i); // 인접 리스트 그래프에 정점 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); // 인접 리스트 그래프에 무작위 간선 추가

}

}

}

print\_adj\_list(gg); // 인접 리스트 그래프 출력

printf("인접 리스트 DFS: ");

for (int i = 0; i < num1; i++) {

if (!visited1[i]) {

dfs\_list(gg, i); // 인접 리스트 그래프에 대한 DFS 수행

}

}

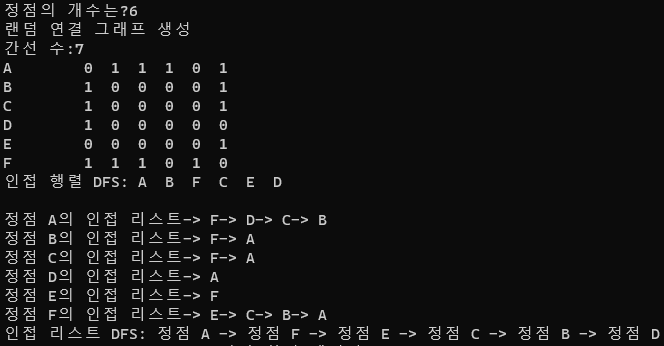
free(g); // 동적 할당된 인접 행렬 그래프 메모리 해제

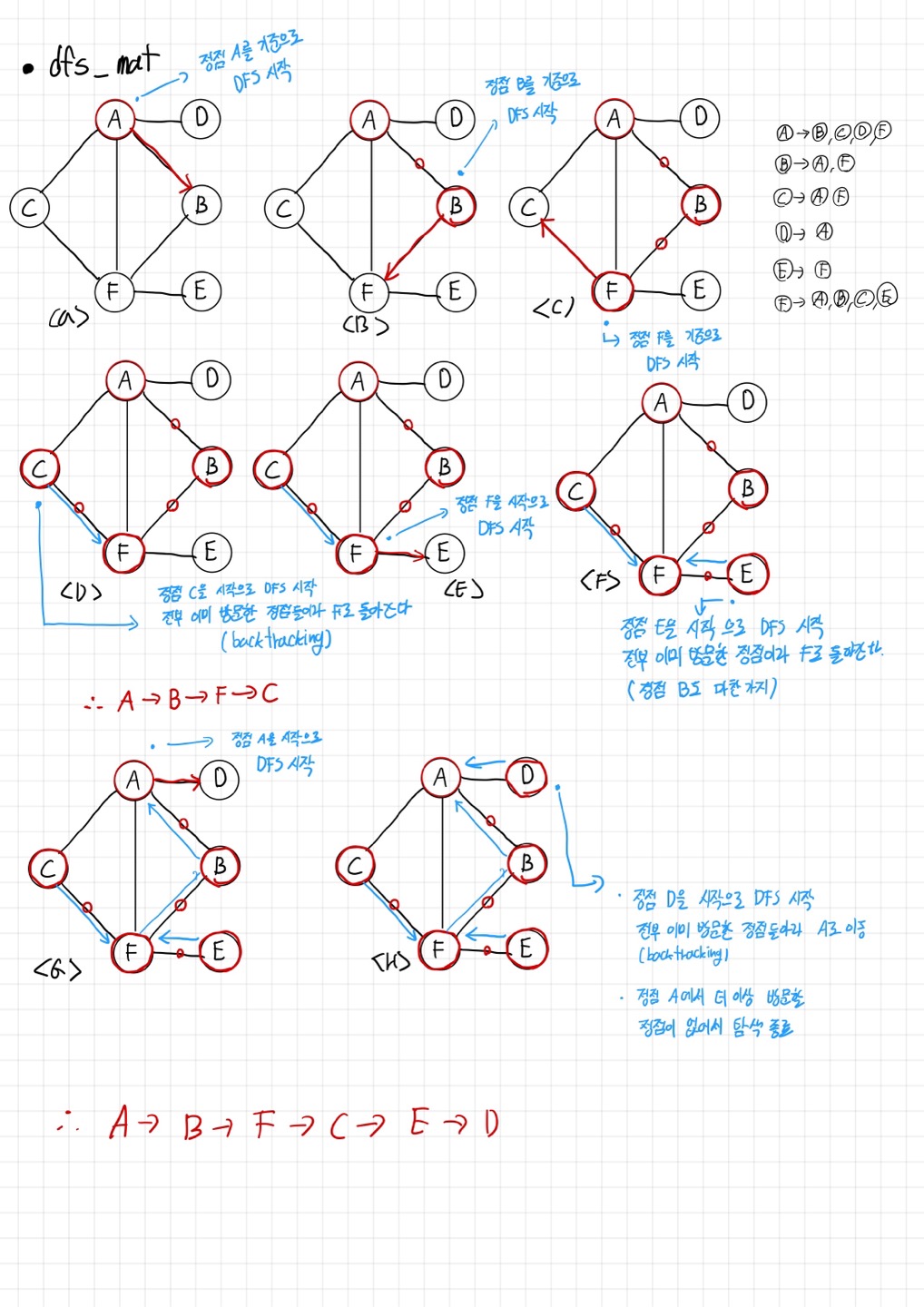
free(gg); // 동적 할당된 인접 리스트 그래프 메모리 해제

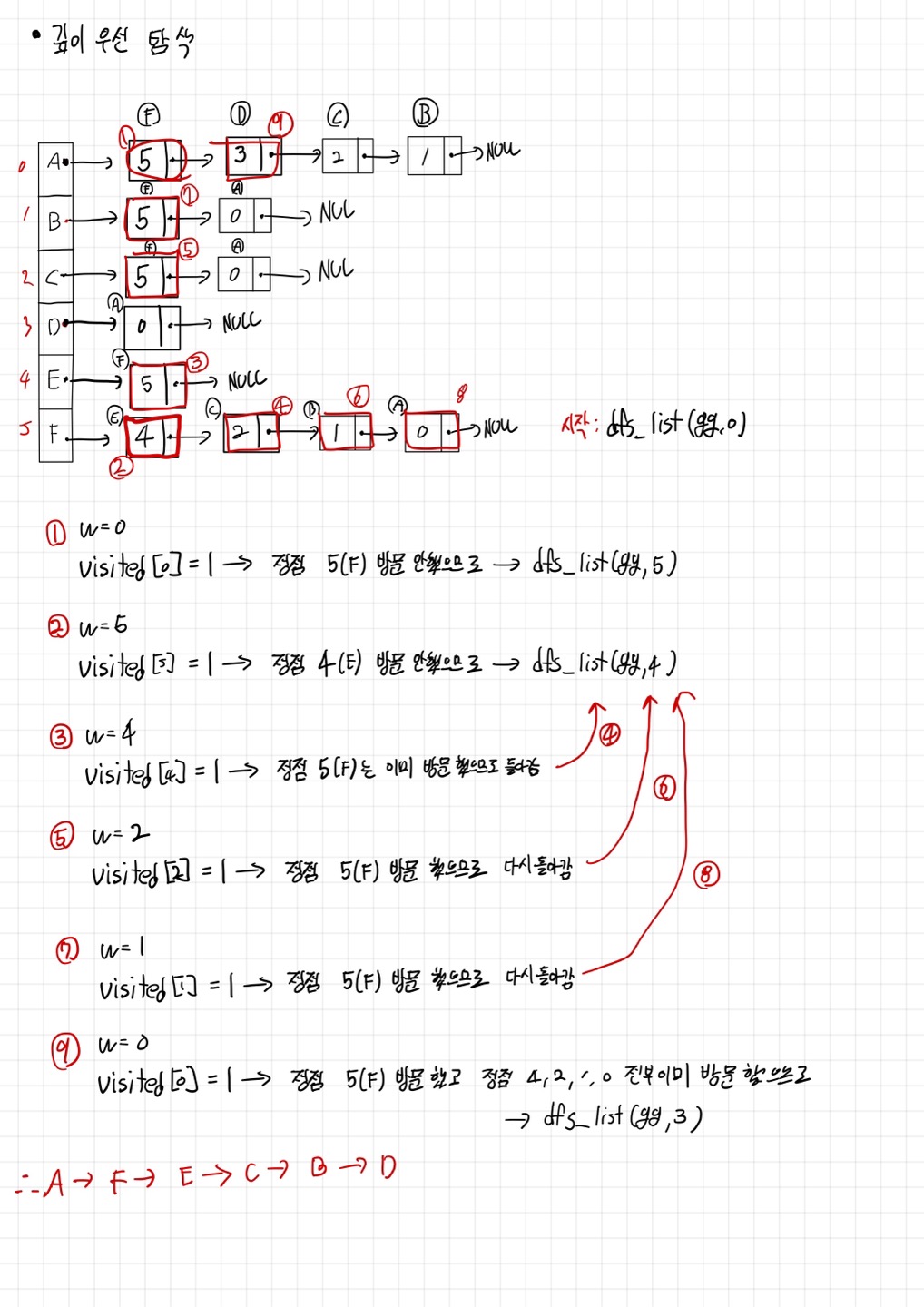
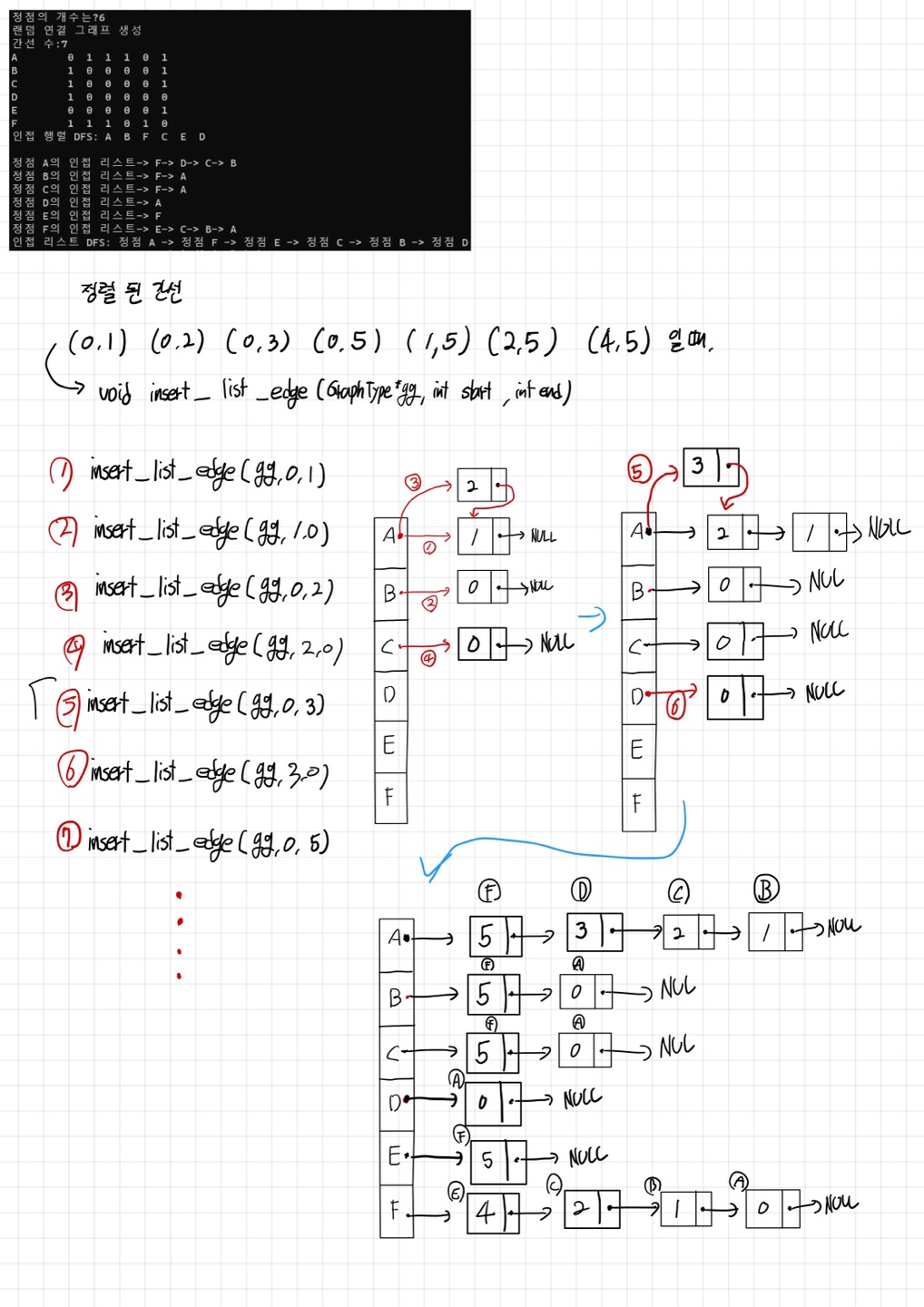
return 0;

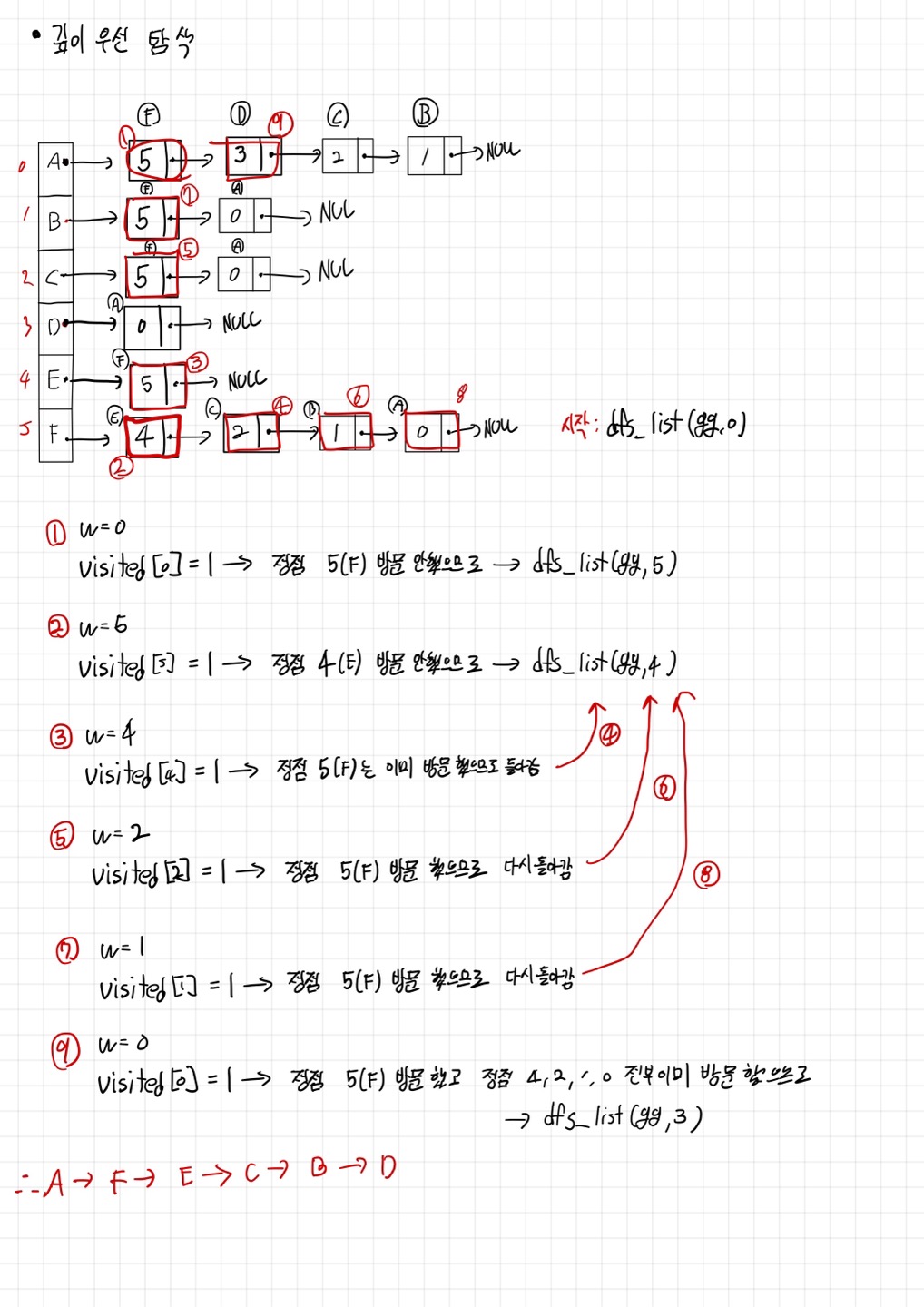
}

1. **실행 결과**

****

****

****

****

1. **설명**

* init()  
  그래프를 초기화 합니다.  
  그래프의 정점의 개수 n을 0으로 설정하고, 인접한 행렬의 모든 원소를 0으로 초기화 합니다
* insert\_vertex()  
  그래프에 정점을 삽입합니다  
  그래프의 정점 개수 n을 증가시킵니다.
* insert\_edge()  
  두 정점 사이에 간선을 삽입합니다  
  인접 행렬에서 start에서 end 로 가는 간선과 end에서 start로 가는 간선을 1로 표시합니다
* print\_adj\_mat()  
  그래프의 인접 행렬을 출력합니다  
  그래프의 인접 행렬을 행과 열에 대한 레이블과 함께 출력합니다.
* dfs\_mat(g, v)  
  깊이 우선 탐색(DFS)을 수행합니다  
  v에서 출발하여 DFS를 수행하고, 방문한 정점을 출력합니다.
* adj\_produce(g, num1, line)  
  그래프에 무작위 간선을 추가합니다  
  num1개의 정점 중에서 무작위로 선택하여 중복된 간선을 피하여, 그래프에 간선을 추가합니다.