**로고, 폰트, 텍스트, 상징이(가) 표시된 사진

자동 생성된 설명**

**[데이터구조2 리포트]**

**이름: 황지상 학번: 20224060**

**과제#2\_Graph\_DFS**

1. **코드**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define MAX\_VERTICES 50 // 최대 정점 개수를 50으로 정의

#define TRUE 1 // 논리적 참을 나타냄

#define FALSE 0 // 논리적 거짓을 나타냄

// 그래프 구조체 정의

typedef struct GraphType {

int n; // 정점의 개수를 나타내는 변수

int adj\_mat[MAX\_VERTICES][MAX\_VERTICES]; // 인접한 행렬을 나타내는 2차원 배열

}GraphType;

// 그래프 초기화 함수 정의

void init(GraphType\* g) {

int r, c;

g->n = 0;

// 인접 행렬의 모든 원소를 0으로 초기화

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

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

g->adj\_mat[r][c] = 0;

}

}

}

// 정점을 삽입하는 함수 정의

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

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

// 정점의 개수가 초대 개수를 초과할 경우 에러 메시지 출력

fprintf(stderr, "그래프: 정점의 개수 초과");

return;

}

g->n++;

}

// 두 정점 사이에 간선을 삽입하는 함수

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

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

// 정점 번호가 유효 범위를 벗어나면 오류 메시지 출력

fprintf(stderr, "그래프: 정점의 번호 오류");

return;

}

g->adj\_mat[start][end] = 1; // start에서 end로 가는 간선을 표시

g->adj\_mat[end][start] = 1; // end에서 start로 가는 간선을 표시

}

// 인접 행렬을 출력하는 함수

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

{

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

printf("%c\t", 'A' + i); // 정점 레이블 출력

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

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

}

printf("\n");

}

}

// 깊이 우선 탐색(DFS)을 수행하는 함수

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

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

int w;

visited[v] = TRUE; // 현재 정점을 반문한 것으로 표기

printf("%c ", 'A' + v); // 현재 정점을 출력

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

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

// 현재 정점과 연결된 정점 중 방문하지 않은 정점이 있다면 재귀적으로 탐색

dfs\_mat(g, w);

}

// 두 정수를 입력으로 받아 같으면 0, 다르면 1을 반환하는 함수

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

if (x == y) {

return 0;

}

return 1;

}

// 그래프 g에 간선을 추가하는 함수 정의

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

srand(time(NULL)); // 난수 생성을 위해 현재 시간을 시드로 사용

// 간선의 두 끝점을 저장할 배열 할당

int\* line\_list\_x = (int\*)malloc(sizeof(int) \* line);

int\* line\_list\_y = (int\*)malloc(sizeof(int) \* line);

//배열 인덱스

int index = 0;

// line 개수 만큼 반복하여 무작위 간선을 생성

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

// 무작위로 정점 선택

int randomx = rand() % num1;

int randomy = rand() % num1;

int chk = 0;

// 같은 간선을 중복해서 생성하지 않도록 선언

if (random\_same(randomx, randomy)) {

// 이미 생성된 간선과 비교하여 중복 확인

for (int j = 0; j < index; j = j++) {

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;

i--; // 중복된 경우 다시 시도

break;

}

}

// chk 변수가 1이 아닌 경우 (중복된 간선이 없는 경우)

if (chk != 1) {

// 두 정점을 오름차순으로 저장하여 중복을 피하기 위한 작업

if (randomx < randomy) {

line\_list\_x[index] = randomx;

line\_list\_y[index] = randomy;

}

else {

line\_list\_x[index] = randomy;

line\_list\_y[index] = randomx;

}

index++; // 중복을 피한 간선을 배열에 저장하고 배열 인덱스 증가

insert\_edge(g, randomx, randomy); // 중복이 없는 간선을 그래프에 추가

}

}

else {

i--; // 같은 정점을 연결하려고 한 경우 다시 시도

}

}

}

int main(void) {

int num1 = 0, line = 0;

int randomx, randomy;

GraphType\* g;

g = (GraphType\*)malloc(sizeof(GraphType));

init(g);

printf("정점의 개수는?");

scanf("%d", &num1);

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

insert\_vertex(g, i); // 입력받은 정점의 개수만큼 그래프에 정점 추가

}

printf("랜덤 연결 그래프 생성\n");

printf("간선 수:");

scanf("%d", &line);

adj\_produce(g, num1, line); // 랜덤 간선을 생성하여 그래프에 추가하는 함수 호출

print\_adj\_mat(g); // 그래프의 인접 행렬을 출력하는 함수 호출

printf("DFS: ");

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

if (!visited[i]) {

dfs\_mat(g, i); // 방문하지 않은 정점부터 깊이 우선 탐색을 수행하는 함수 호출

}

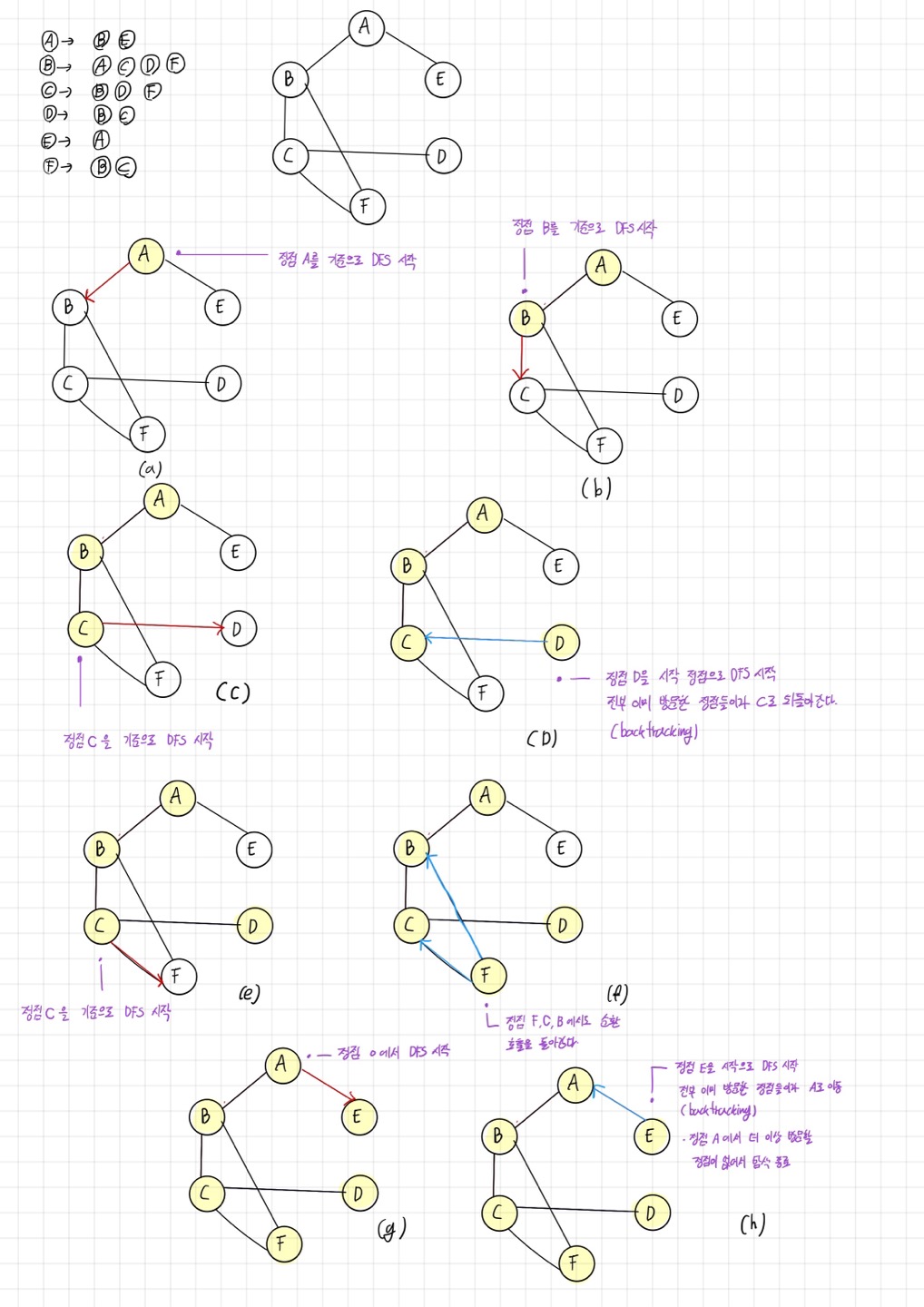
}

free(g);

}

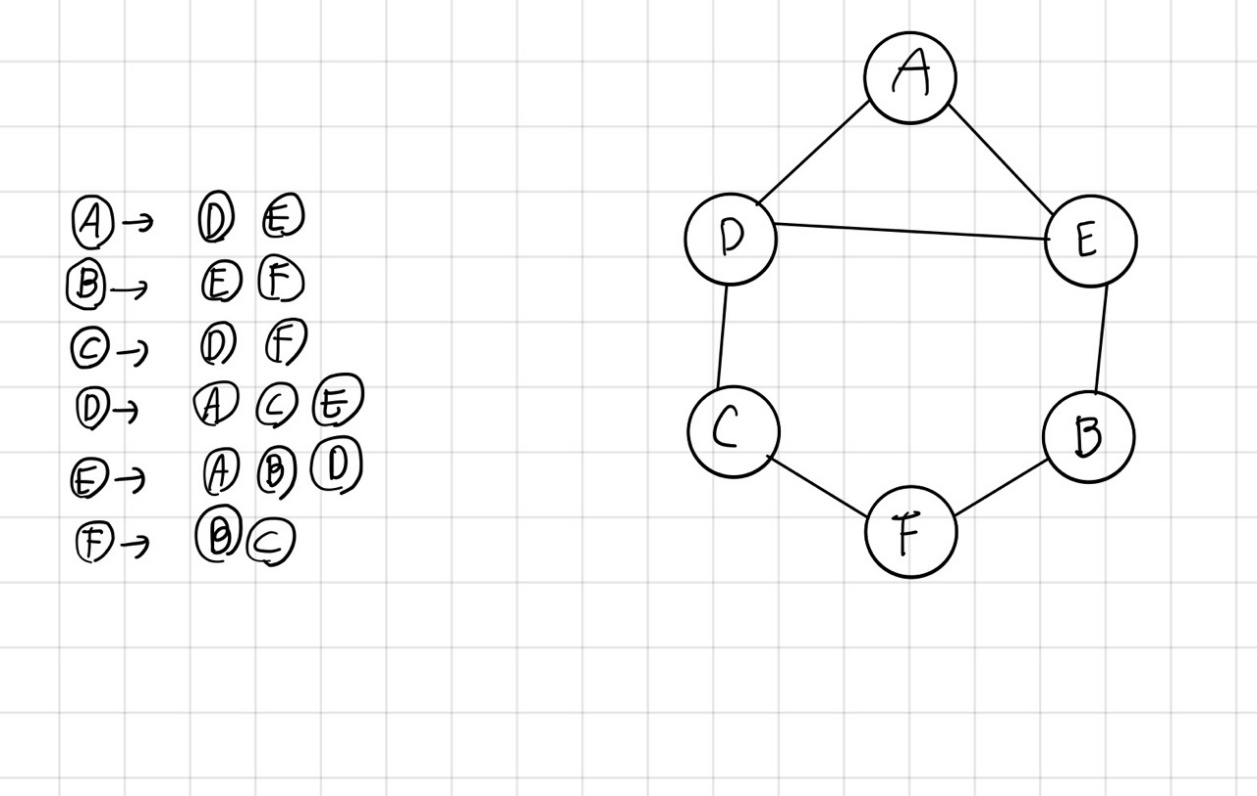
1. 텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

   자동 생성된 설명**실행 결과**

****

**텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명**

****

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.