**알고리즘 HW2**

최지웅 교수님

Floyd’s Algorithm for Shortest Paths

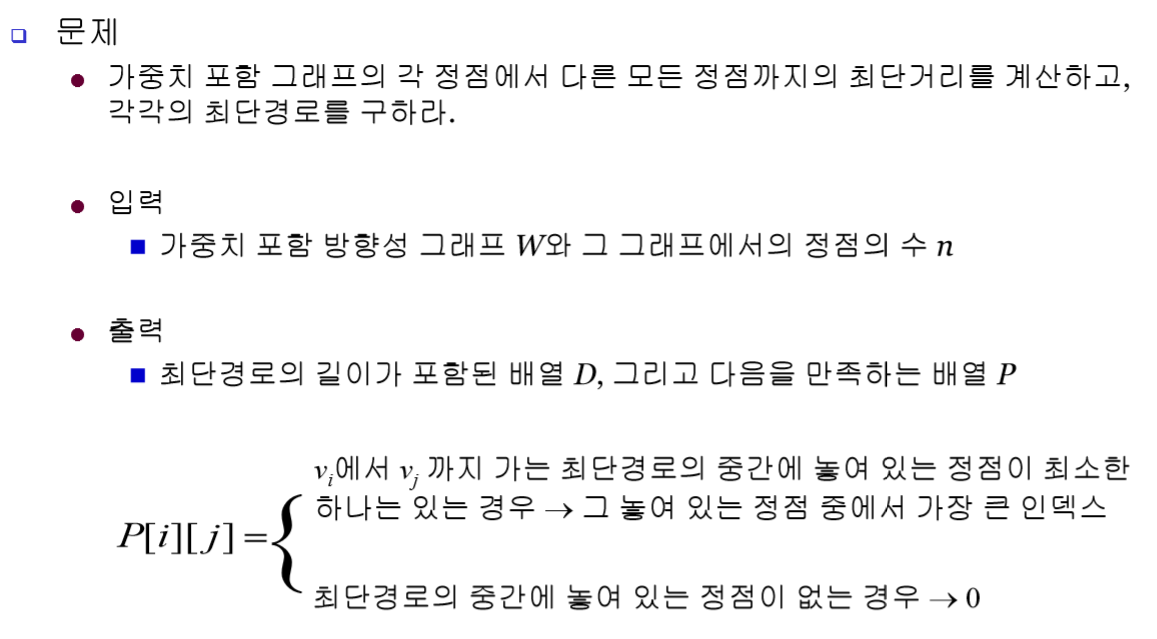
/ 3장 Dynamic Programming

**20150439 소혜빈**

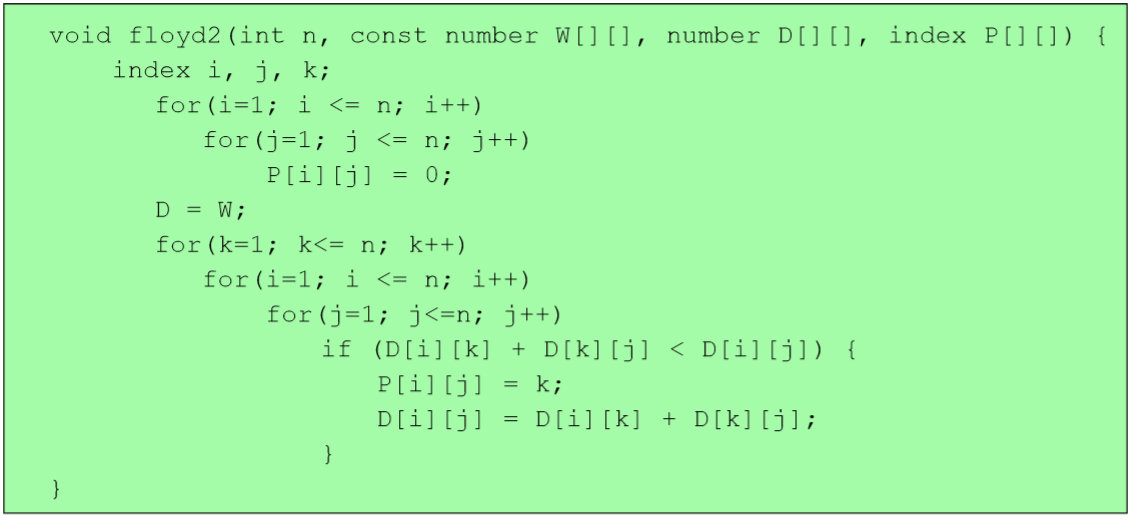
**목차**

* Problem
* Class
* 반복문
* 무한대
* Path
* 결과 확인
* 손풀이

**Problem**



**Algorithm :**



*출처 : 숭실대 스마트캠퍼스 알고리즘 강의자료*

무한대(두 정점이 연결되어 있지 않음)는 ‘int type의 최댓값/2’ 로 정의한다.

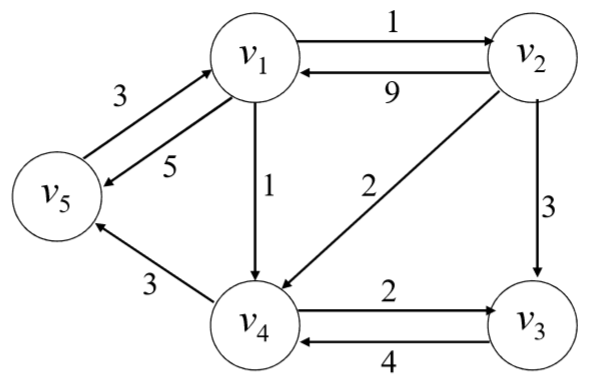
#define INFINITY INT\_MAX/2

**구현 언어 :** C++

**컴파일러 :** gcc-g++

**Problem**

**Sample value 1 :**

****

*출처 : Figure 3.2/ Foundation of Algorithms using C++ Pseudocode 3th*

W[5][5] = {

{0,1,INFINITY,1,5},

{9,0,3,2,INFINITY},

{INFINITY,INFINITY,0,4,INFINITY},

{INFINITY,INFINITY,2,0,3},

{3,INFINITY,INFINITY,INFINITY,0}

};

**Sample value 2 :**

W[6][6] = {

{0,1,INFINITY,INFINITY,2,INFINITY},

{INFINITY,0,14,9,3,INFINITY},

{INFINITY,INFINITY,0,2,INFINITY,INFINITY},

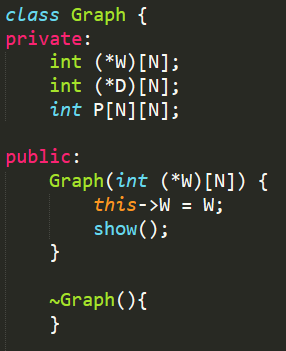
{8,INFINITY,11,0,INFINITY,INFINITY},

{3,INFINITY,INFINITY,INFINITY,0,1},

{5,INFINITY,6,INFINITY,1,0}

};

Class



Graph 클래스를 만들어 인접행렬 값을 가진 이차원 배열을 main함수에서 받아 생성하고,









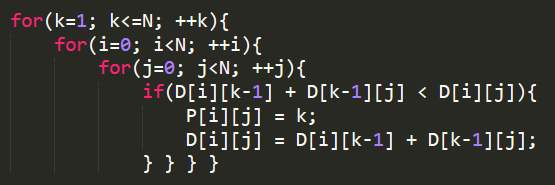


위 함수들을 내부에 구현 하였다.

반복문

void floyd() {

…



}

k(들렸다 가는 vertex, P의 원소)는 1~n까지 증가한다고 나타낼 때,

배열의 index는 0~n-1 까지 존재하기 때문에,

k가 배열의 index로 사용 될 때에는 -1 해줘야 한다.

무한대





<climits> 라이브러리의 INT\_MAX 를 INFINITY 값으로 사용하려고 했으나,

*참고: http://www.cplusplus.com/reference/climits/*

D(k-1)[i][k]+D(k-1)[k][j] 와 같이 + 연산에서 INT\_MAX 에 양의 값을 더하게 되면 overflow가 발생하여 정상적인 계산이 이루어 질 수 없어서 적당히 2로 나눠준 값을 무한대로 정의해서 사용하였다.

또, 행렬 출력 시 INFINITY 라고 요소를 출력하는 것은 모양이 예쁘지 않아서

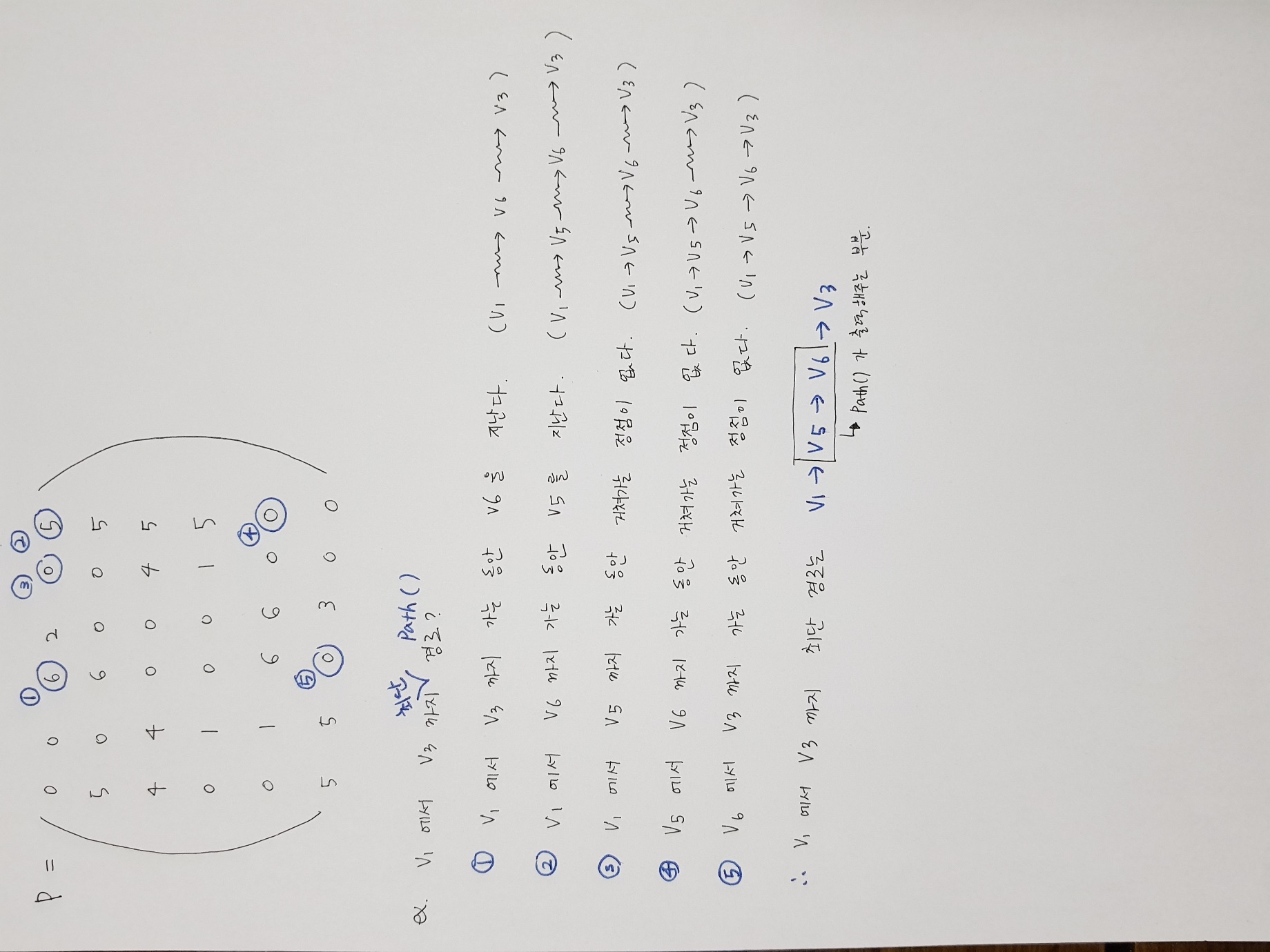
무한대 값의 경우 ‘E’ 로 출력하도록 하였다.



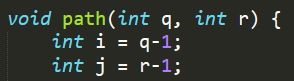
*값이 너무 커서 error 라는 의미.*

*Infinity의 I를 사용할까 했으나, 숫자 1과 혼동되기 쉬워 보여 E로 결정하였다.*

Path



파라미터 입력은 정점 번호로 받고, 함수 내에서 배열의 index로 사용될 때에는 각각 -1한 값을 i, j에 넣어 사용하였다.

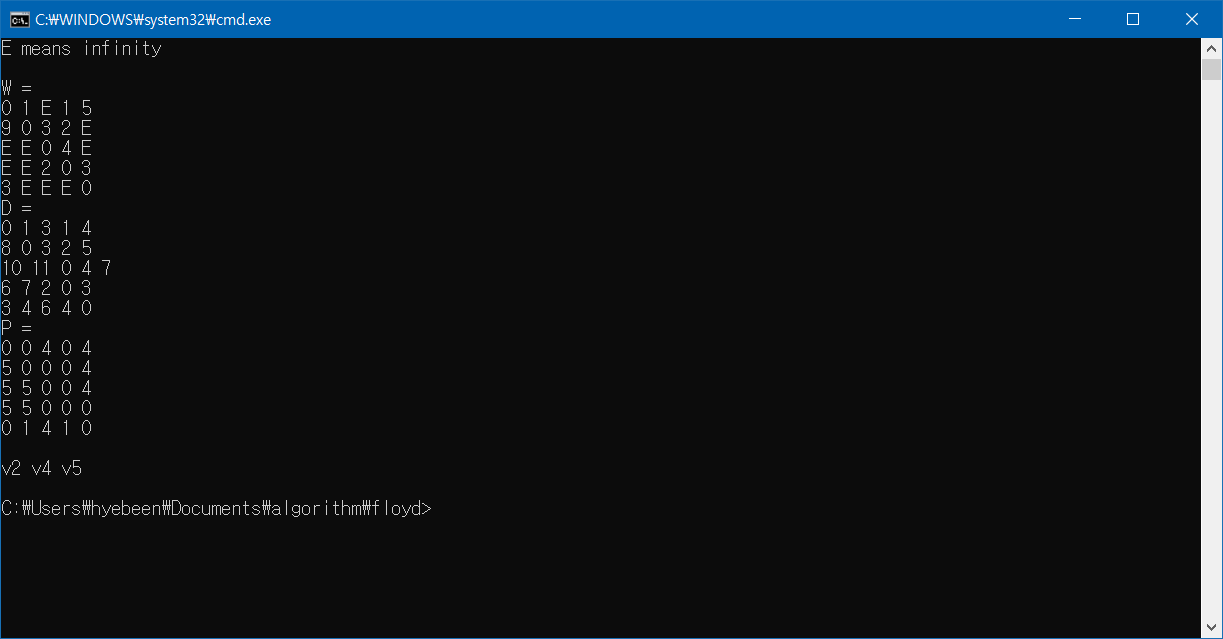


…

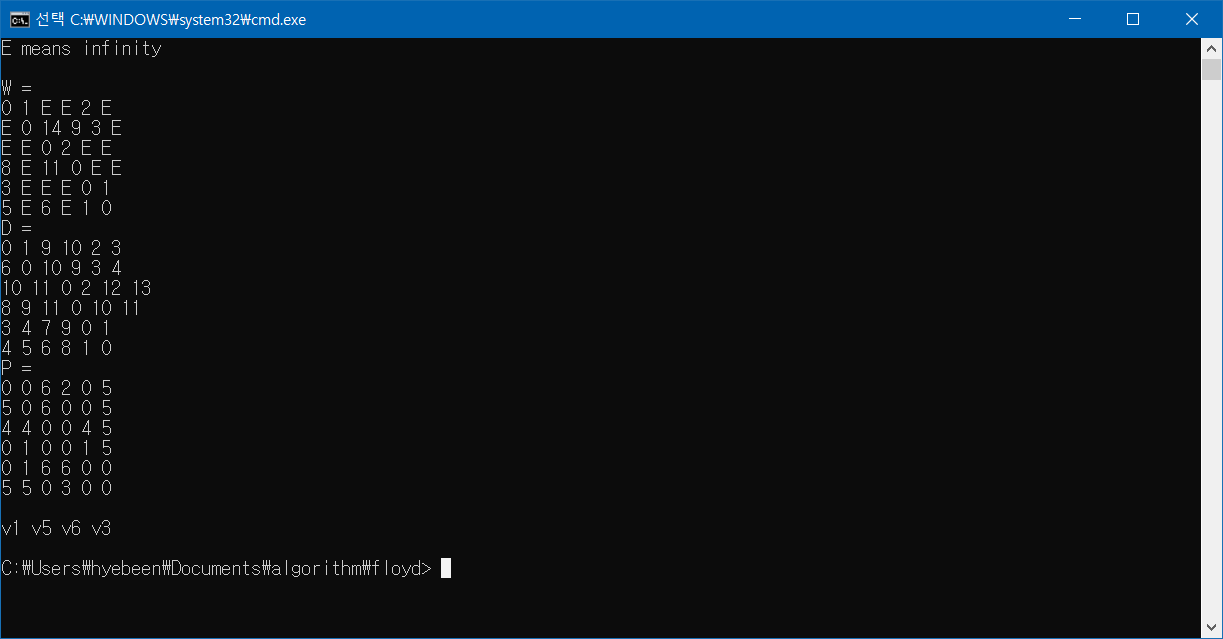
}

결과 확인

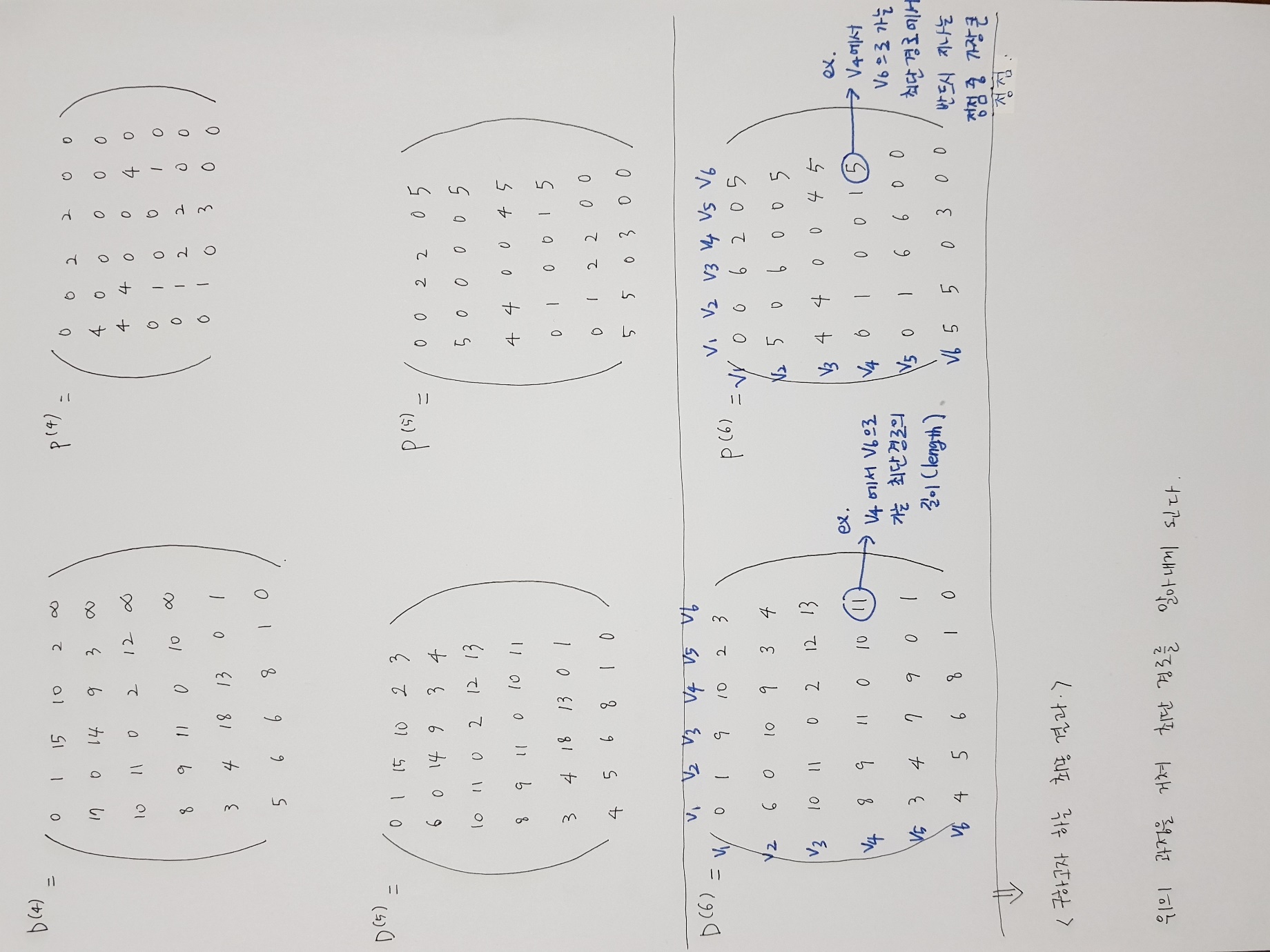
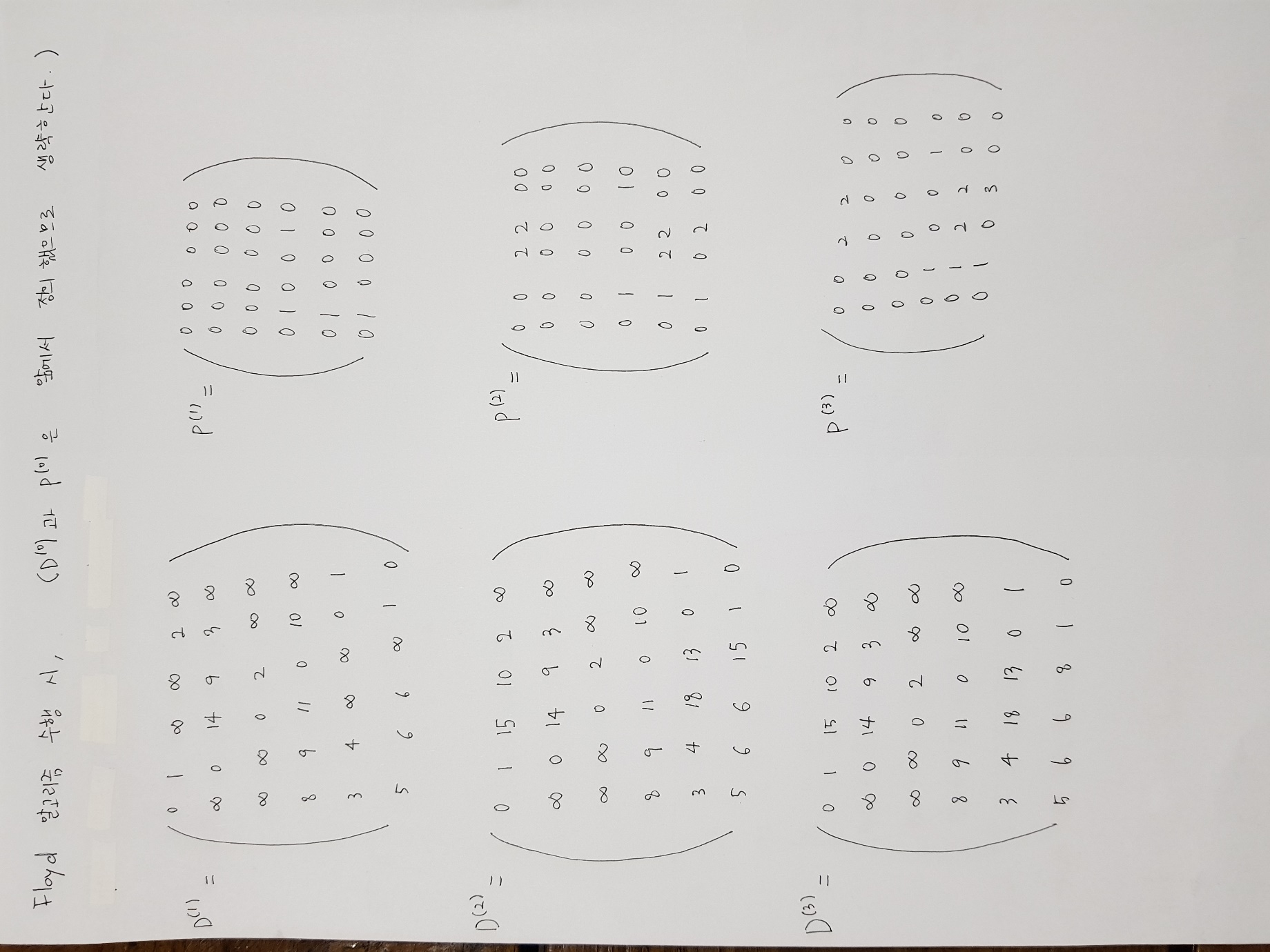
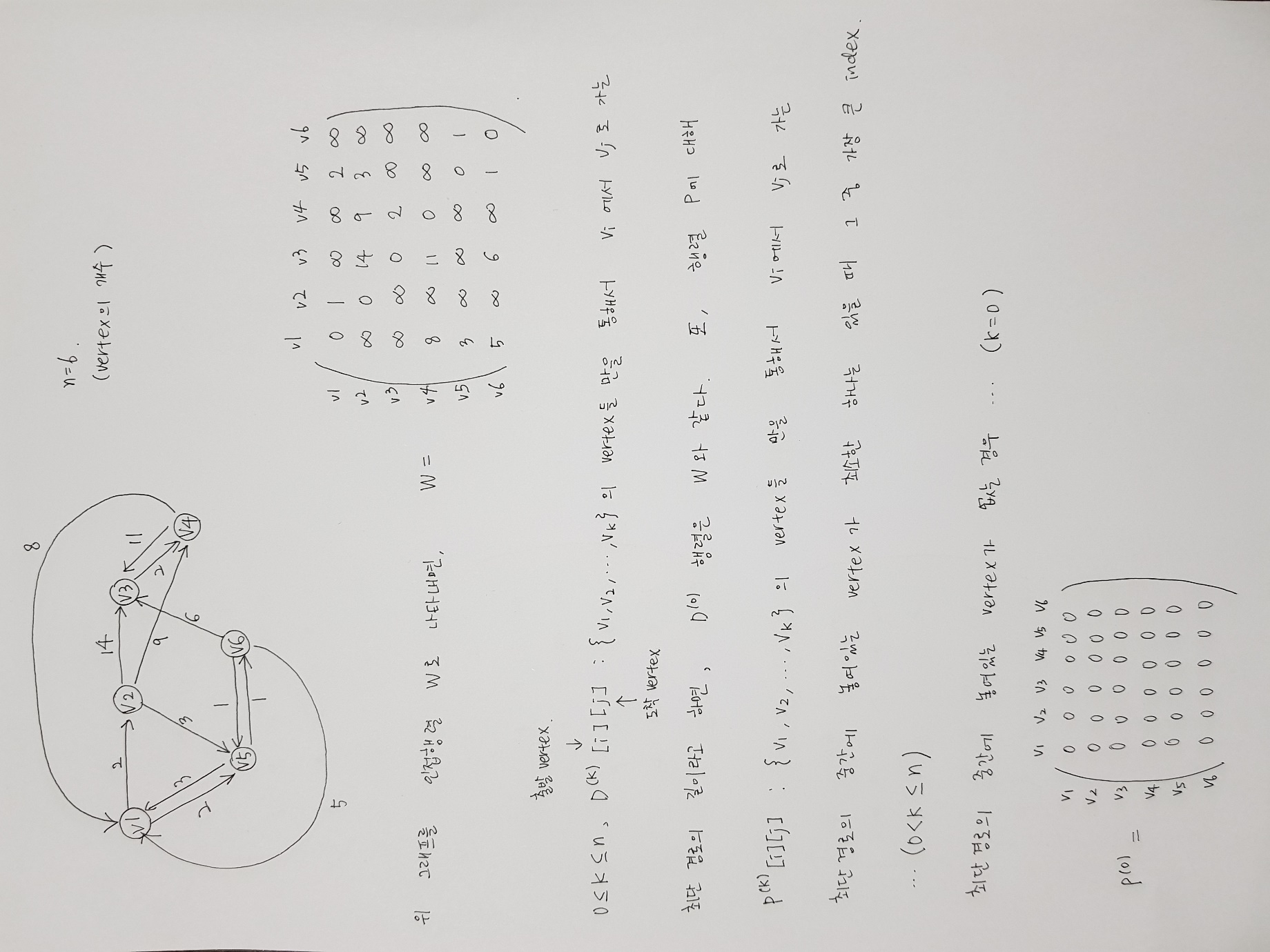
**Sample Value 1**



**Sample Value 2**



**Sample Value 2 손 풀이**



결과는 같다.

배열의 요소를 하나하나 구하는 과정을 모두 보여주기엔 너무 많아서 k와 i값을 골라 가장 안쪽 loop의(j) 계산 과정 몇 개만 보인다.

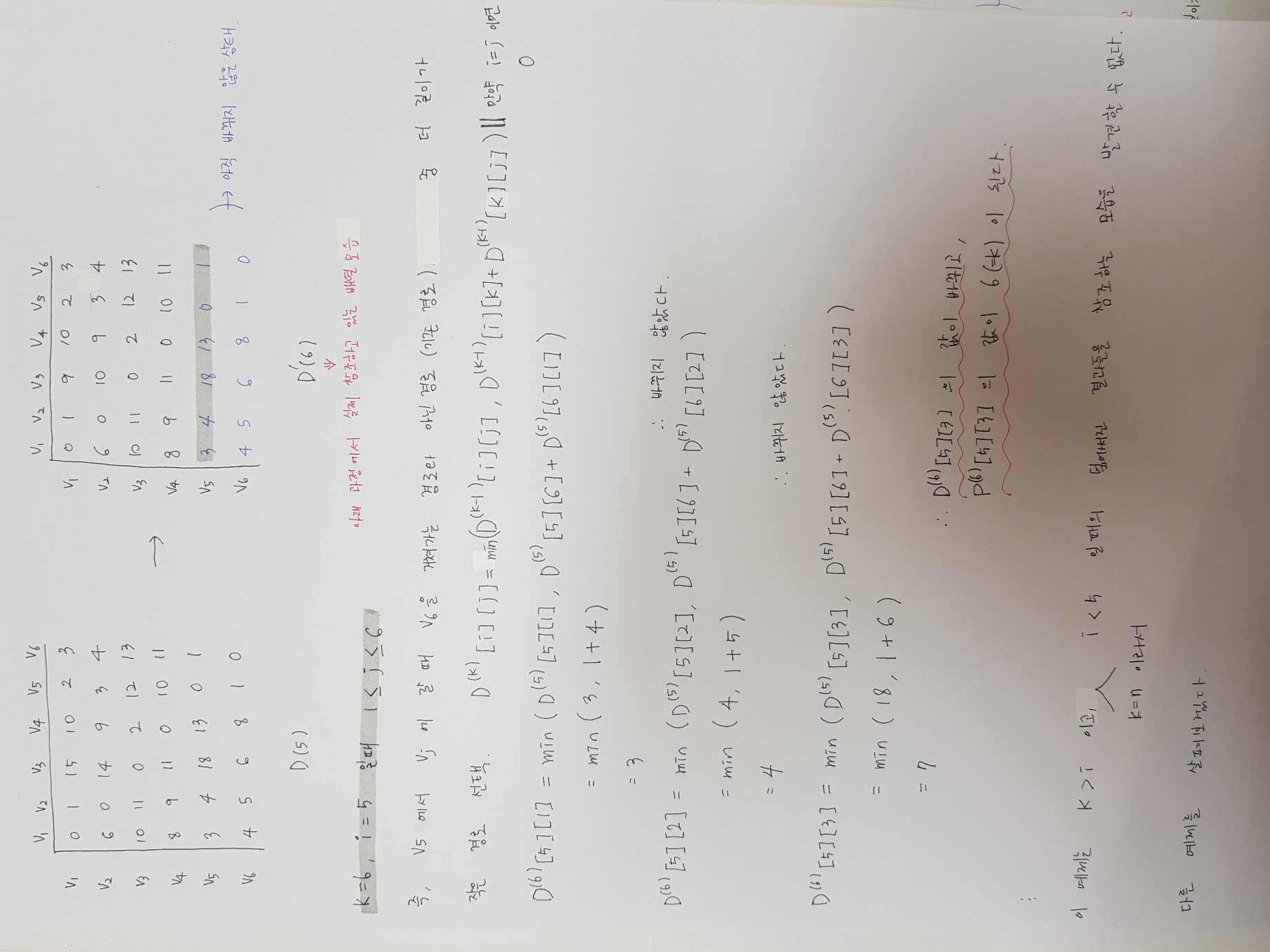
또, 수업자료에는 알고리즘 설명 부분에 i, j의 값을 고정시켜 놓은 상태에서 k값을 변화시키면서 계산 (거쳐가는 vertex의 값을 하나씩 증가시켜가며 계산) 하고 있으나,

구현 코드에서는 k가 가장 바깥 쪽 for문에 나타나며 k값을 고정시켜 놓은 상태에서 i, j값을 변화시켜가며 계산 (거쳐가는 vertex의 값을 고정해두고 출발 vertex와 도착 vertex의 값을 하나씩 증가시켜가며 계산) 하고 있기 때문에 **프로그램 동작순서**에 맞춰 계산하였다.

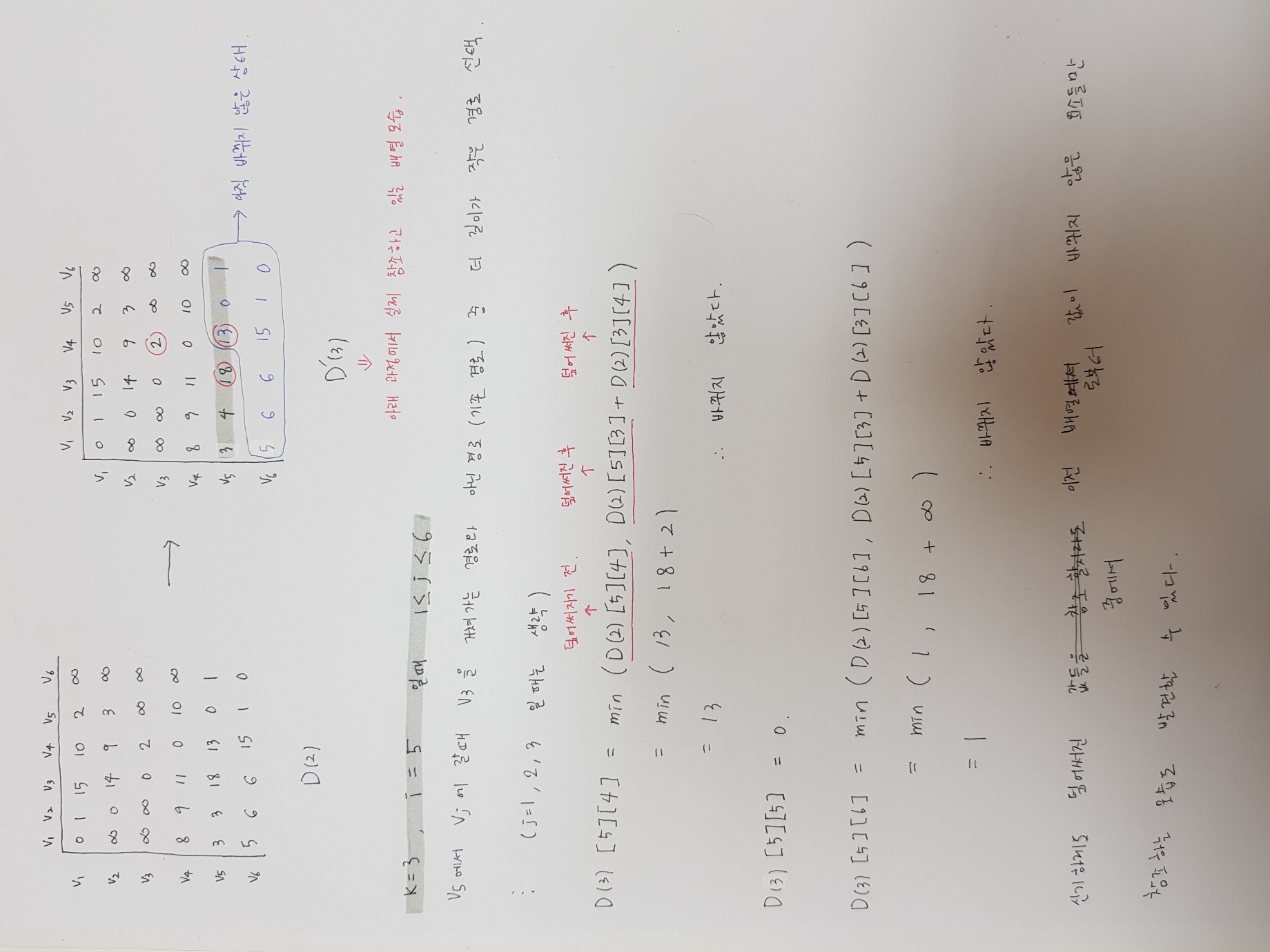
마찬가지로 실제 프로그램에서는 배열이 덮어써진다는 점을 고려하여 계산 한다.

**단**, index의 경우 사이즈가 n인 배열에서는 실제로 0~n-1 까지의 인덱스를 갖지만 그래프의 vertex임을 강조하고 단순하게 표현하기 위해 1~n 까지의 index를 사용한다고 가정한다.

실제 코드에서는 [6p](#floyd) , [8p](#path) 와 같이 index를 사용하였다.



위 예시는 k > i 이고, 또 하필 k = n 이기 때문에 계산 과정에서 배열에 덮어씌워진 값들이 사용되지 않는다. j = 1,2,3 까지만 계산하고 4,5,6 은 생략하고 또 다른 예시를 아래에서 보이겠다.



배열에 이미 덮어씌워진 값을 계산에 사용할 때, 선택된 덮어씌워진 값들이 모두 이전 버전배열(D(2))에서 바뀌지 않았던 값들만 선택되는 모습을 발견할 수 있다.