Code 설명

* Graph.txt에 맨 첫 줄에는 순서대로 ‘vertex 개수’, ‘시작점’, ‘목적지’, ‘경유지’를 vertex number로 구분하여 기입하였고 다음 줄 부터는 vertex간의 edge 거리를 6 X 6 배열로 표기를 해줬다. 이때, vertex 간 거리는 항상 양수이며 서로 연결 되어 있지 않은 경우 무한대로 인식을 하도록 매우 큰 수 100000000을 edge거리로 입력하였다.
* Dijkstra Algorithm을 사용하여 Shortest path 를 구하였다. 이를 사용하기 위하여 각 vertex 방문 여부를 확인하기 위하여 bool v[6]을 변수로 사용하여 각 vertex 방문 여부 표시를 해줬다. 또한 시작점에서 각 vertex 까지의 최단 거리가 얼마인지를 표시하기 위하여 int d[6] 변수를 사용하였고, Dijkstra Algorithm을 사용하여 매 단계별로 d[i]의 값을 갱신하였다.

>small 함수

매 과정에서 방문하지 않은 vertex 중 가장 작은 d[i]값을 탐색하여 가장 작은 값을 가지고 있는 vertex의 i 값을 index로 가져온다. 결과로 index 값을 반환한다.

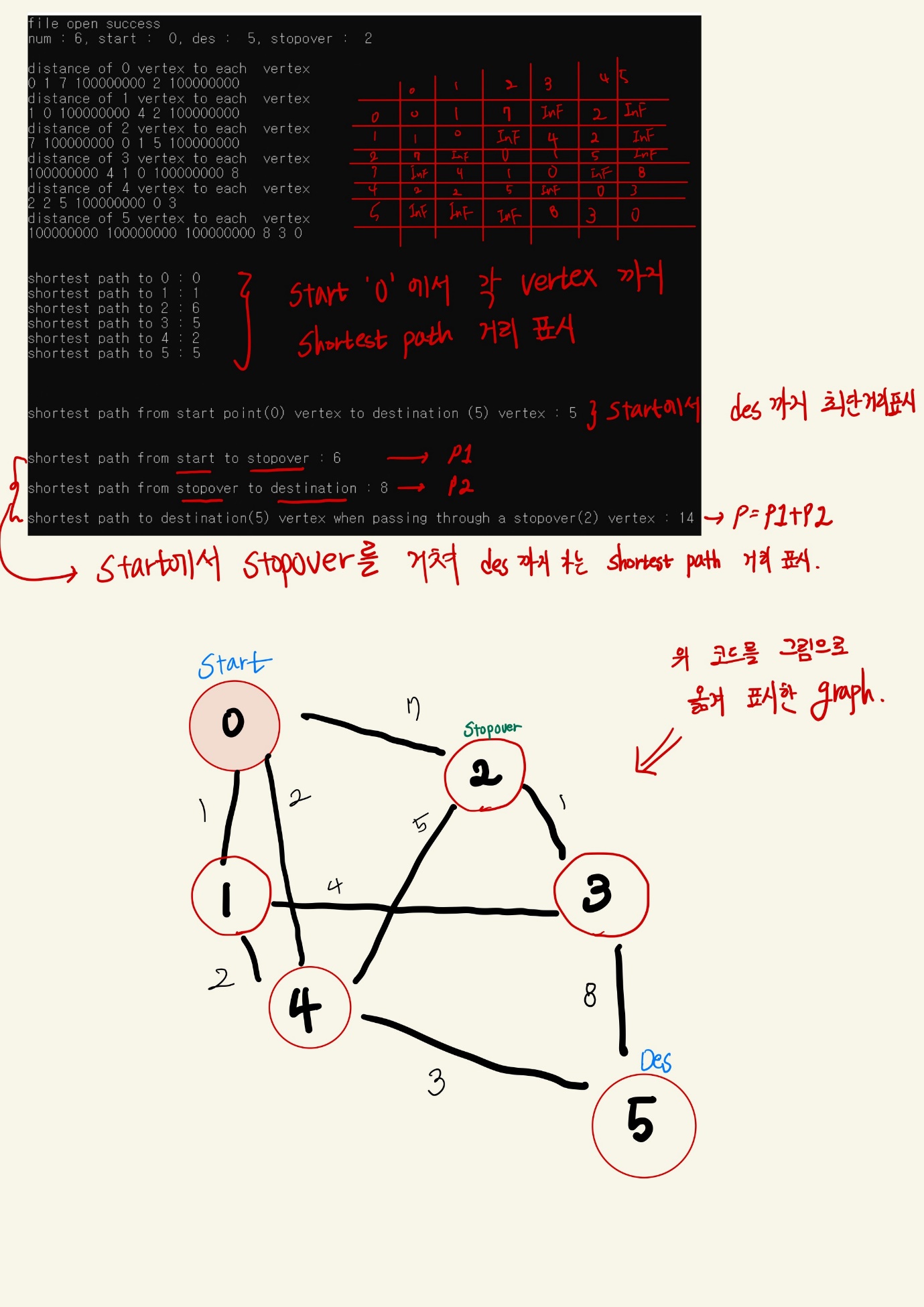
>Dijk 함수

start값을 입력으로 받아 각 vertex까지의 shortest path 를 구하는 함수이다. d[i]를 시작점에서 각 vertex까지의 거리로 초기화한 후, start vertex 방문을 표시하기 위해 v[start]함수의 값을 1로 표시한다. 이후 방문하지 않은 vertex들 중 가장 짧은 거리를 찾기 위해 small함수를 사용하여 small 함수의 출력값인 index값을 now 변수에 저장한다. now값의 vertex를 방문했다는 것을 표시하기 위해 v[now]에 1을 넣어준다. 이후 방문하지 않은 vertex (v[k] == 0)중 now vertex까지의 거리와 now에서 k까지의 거리를 더한 값 d[now] + edge[now][k]과 기존 k까지의 최단 거리의 값 d[k]의 대소 관계를 비교하여 d[k]값이 더 크다면 d[k]값을 d[now] + edge[now][k]으로 갱신하여 준다. 해당 과정을 총 num-2번 반복하여 주면 모든 vertex에 대하여 d[i]의 값이 갱신된다.

>Dijk\_stopover 함수

기본적으로는 위 Dijk함수 Algorithm을 그대로 사용한다. Start, stopover, des값을 입력으로 받아 시작점에서 경유지를 거쳐 도착점까지의 최단 경로를 표시해준다. p1, p2, p 변수를 사용하여 p1은 시작점에서 경유지까지의 거리, p2는 경유지에서 도착점까지의 거리를 의미한다. 이때, 시작점에서 경유지를 거쳐 도착점까지의 최단거리 p = p1 + p2가 된다. 해당 과제에서는 경유지를 1개로 설정하였지만, 경유지가 2개 이상이라면 해당 부분 코드가 좀 더 복잡해질 것이다. 이때 경유지를 모두 거쳐야 하는지, 경유지를 하나만 거쳐도 되는지에 따라서 또 달라질 것이다. 거기다 경유지가 3개라면 경유지를 몇 개 거치는지 여부에 따라서 코드가 바뀔 것이다. 만약 경유지가 2개이고, 경유지를 하나만 거쳐도 된다면, 본 Dijk\_stopover 함수의 입력에 각 경유지를 입력하여 나온 결과를 비교하여 최솟값을 출력하는 함수를 구현하면 된다.

 다음은 컴파일 후 콘솔화면에서 실제로 구현해본 결과이다.



텍스트이(가) 표시된 사진

자동 생성된 설명