INHA UNIVERSITY

알고리즘 설계 HW #3

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.
2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.
3. 나는 보고서의 내용을 조작하지 않겠습니다.
4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.
5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2020년 6월 14일

학부 정보통신공학과

학년 3학년

성명 김소원

학번 12181748

1. 개요

*** 아래와 같은 graph 알고리즘을 작성하기.**

(1) BFS, DFS 작성.

(a) "알고리즘설계_2020_Graph_6월9일_주석.pdf"의 21페이지의 그래프를 대상으로 정상 동작함을 보이기

- 이 때 방문 되는 노드만 출력하지 말고 간선도 함께 표시 –

(b) 노드의 수가 20개 이상인 그래프를 임의로 정의하고 정상 동작함을 보여라.

(2) Bellman-ford 알고리즘을 작성하고 정상 동작함을 보이기.

(a) 싸이클 감지 알고리즘을 어떻게 작성했는지 자세히 설명할 것

(b) "알고리즘설계_2020_Graph_6월9일_주석.pdf"의 69,72페이지의 그래프를 대상으로 각각 정상 동작함을 보이기

(c) 노드의 수가 20개 이상인 그래프를 임의로 정의하고 정상 동작함을 보여라.

2. 상세 설계내용

(1) BFS, DFS 작성

BFS(Breadth First Search), 너비 우선 탐색 알고리즘은 거리가 가까운 것 먼저 탐색하는 알고리즘으로 코드를 짤 때 큐를 이용하였다. 먼저 큐에서 하나의 노드를 꺼낸 후 해당 노드에 연결된 노드 중 방문하지 않은 노드를 방문하고 차례대로 큐에 삽입하는 과정을 생각해보고 이 과정을 반복하도록 설계하였다.

DFS(Depth First Search), 깊이 우선 알고리즘은 스택의 최상단 노드를 확인 후 최상단 노드에게 방문하지 않은 인접 노드가 있다면 그 노드를 스택에 넣고 방문 처리하는 과정을 생각해보았다. 만약 방문하지 않은 인접 노드가 없다면 스택에서 최상단 노드를 빼도록 하며 이 과정을 반복하도록 설계하였다.

나는 BFS알고리즘과 DFS 알고리즘의 결과가 한번에 나오도록 하나의 소스 파일을 이용하였다. 먼저 큐를 사용해야 하므로 #include<queue>를 하였으며 최대값을 임의로 설정하였다. 그래프의 값을 보이기위해 그래프의 배열과 노드 개수만큼 방문하는 visit 배열을 만들었으며 node는 노드를, line은 간선, f는 first의 첫 글자 f를 따서 각각을 선언하였다. BFS 함수를 먼저 본다면 처음 노드를 방문하면 그 노드를 큐에 넣어야 하므로 q.push를 사용한다. 그 다음으로 방문한 노드가 들어오면 이전 노드는 나가므로 pop을 해준다. 방문하지 않은 노드도 찾아야 하는데, i가 노드의 갯수만큼 증가할 때 노드 f와 i가 만나고, 노드 i는 방문하지 않으면 노드 i의 값을 큐에 넣어야 한다.

그 다음으로 DFS 함수를 본다면 가장 먼저 노드를 방문해야 한다. f는 노드를 의미하며 i가 노드의 개수만큼 증가하는 동안 다음 노드를 방문했으면 Dfs 함수를 호출한다. 메인 함수에서는 먼저 node는 노드 개수, line 은 간선 갯수, f는 시작 노드라 정하고 각각을 입력 받는다. memset함수는 graph와 visit의 값을 모두 지정하기 위해 사용하였다. 그 다음 과제에 나와있는 방문 되는 노드만 출력하지 말고 간선도 함께 표시하기 위해 연결된 두 노드들을 다 입력 받도록 하였다. 이렇게 간선과 노드들을 입력하고 나면 BFS와 DFS를 이용한 결과값이 나오게 된다.

(2) Bellman-ford 알고리즘 작성

벨만 포드알고리즘은 다익스트라 알고리즘과 마찬가지로 시작점을 정한 후 다른 노드들까지 최단 경로를 구하는 것이다. 이때 간선 값이 음수도 가능한데, 일부 시작점에서 도착점까지 갈 때 경로 중간에 존재하는 음의 싸이클들이 계속 반복되면 무한한 음수의 값을 가지게 된다. 이는 최단 경로를 구하는데 있어 문제가 되기 때문에 음의 값을 가지는 순환이 없는 경우로 생각하고 설계하였다.

구조체를 이용하여 설계하였는데 먼저 노드 구조체를 만들었다. 정점, 방문할 노드, 가중치를 vertex, visit, weight라 선언하였으며 그 다음으로 graph 구조체를 선언하였다. V는 정점의 갯수이며 line은 간선의 갯수이다. 최단 경로 함수는 제일 마지막에 vertex 최단 경로를 출력 받기 위한 함수로 노드의 최단 경로를 출력한다. 벨만 포드 함수는 먼저 V를 graph의 V로, line은 graph의 line으로 지정한다. 그 다음 Distance[V]를 선언해주어야 하는데 그냥 Distance[V]라고 선언 할 시 상수가 없다고 에러가 떠 malloc를 사용하였다. 가장 먼저 i가 노드의 개수만큼 증가할 때 거리 값을 무한대로 지정해준다. 처음 시작점은 0으로 설정하였다. i가 V-1만큼 증가하는 동안 j는 간선의 개수만큼 for 루프를 돌고 그 안에서 시작점이 무한한 값이 아니면 다음 노드로 방문해준다. 이때 i가 V-1까지만 가는 이유는 최단 경로를 구해야 하는 것인데 최단 경로는 같은 노드를 두 번 방문할 필요는 없기 때문에 간선의 개수를 V에서 1개 빼주는 것이다. 만약 시작점부터 최단거리에 가중치를 더한 값이 도착 노드의 가중치보다 작을 때 작은 값으로 바꿔주어야 한다.

음의 싸이클이 반복되는 것을 주의해야 한다 그랬는데 그 다음 코드가 바로 음의 싸이클을 출력하는 값이다. 음의 가중치가 경로사이에 있으면 값이 무한히 작아지는 경우가 생기는데 그 경우를 방지하는 코드를 짜보았다. i가 간선의 개수만큼 증가하는 for루프를 돌 때 시작점이 무한한 값이 아닌 경우엔 다음 노드로 방문한다. 만약 현재 노드에서의 최단 거리와 가중치를 더한 값이 계속하여 작아지는 경우 음의 싸이클을 출력하도록 하였다. 이것이 2-(a)에 관하여 설명한 답이다.

마지막으로 메인 함수를 본다면 노드의 개수, 간선의 개수, 시작점을 각각 V, line, F 로 선언하였

다. 먼저 노드의 개수를 입력 받고 그 다음 간선의 개수, 제일 처음 시작하는 노드를 입력 받는다. 함수 공간을 할당해준 후 시작점, 도착점, 가중점을 간선의 개수만큼 차례대로 입력 받는다. 예를 들어 3 5 2를 입력하면 3은 시작점 5는 도착점 2는 가중점으로 3에서 5로 가는 값이 2라는 의미가 된다. 이렇게 값들을 입력 받으면 마지막으로 vertex의 최단 경로가 출력된다.

3. 실행 화면

1. BFS, DFS.cpp

12181748 김소원.cpp

Project50 (전역 범위)

```cpp
#include <iostream>
#include <queue> //큐 사용
#include <cstring>

#define MAX 101 //임의로 설정

using namespace std;

int graph[MAX][MAX]; //그래프
int visit[MAX]; //노드 갯수만큼 방문
queue<int> q;
int node, line, f; //각각 노드, 간선, 시작점


void Bfs(int f) { //BFS 함수
    visit[f] = 1; //노드를 방문하면
    q.push(f); //push해줌

    while (!q.empty()) {
        f = q.front(); // 처음 노드
        q.pop(); //pop해줌

        cout << f << " "; //bfs는 pop으로 이동
        for (int i = 1; i <= node; i++) { //방문하지 않은 연결된 노드 찾기
            if (graph[f][i] && !visit[i]) { //정점f와 i가 만나고,정점i는 방문하지 않음
                q.push(i); //노드 방문했으면 push
                visit[i] = 1; //큐에 넣음
            }
        }
    }
}

void Dfs(int f) { //DFS 함수
    cout << f << " ";
    visit[f] = 1; //노드를 방문함
    for (int i = 1; i <= node; i++) { //i가 노드의 갯수만큼 증가할 때
        if (graph[f][i] && !visit[i]) { //노드 방문했으면
            Dfs(i); //Dfs
        }
    }
}
```

```
42
43  int main(void) {
44
45      cout << "노드 갯수: ";
46      cin >> node;
47
48      cout << "간선 갯수: ";
49      cin >> line;
50
51      cout << "시작 노드(정점): ";
52      cin >> f;
53      cout << endl;
54
55      memset (graph, 0, sizeof(graph)); //memset 함수로 graph 값 모두 지정
56      memset(visit, 0, sizeof(visit)); //visit 값 모두 지정
57
58      cout << "연결된 두 노드들(간선) " << endl;
59      for (int i = 0; i < line; i++) {
60          int a, b; //간선을 연결
61          cout << "=>";
62          cin >> a >> b;
63
64          graph[a][b] = 1; //두 노드가 연결되어있음
65          graph[b][a] = 1; //두 노드 연결
66      }
67
68      cout << endl;
69      cout << "BFS 결과: ";
70      Bfs(f); //Bfs 함수
71      cout << "₩n" << endl;
72
73      memset(visit, 0, sizeof(visit)); //visit 값 모두 지정
74
75      cout << "DFS 결과: ";
76      Dfs(f); //Dfs 함수
77      cout << "₩n" << endl;
78      return 0;
79  }
```

2. BellmanFord.cpp

12181748 김소원 bfs,dfs.cpp | 12181748 김소원 BellmanFord.cpp | 12181748 김소원 끝.cpp

Project50 | (전역 범위)

```cpp
#include <iostream>
#include <string.h>
#include <limits.h>
#include <stdlib.h>
#include<vector>

using namespace std;

struct Node  //노드 구조체
{
    int vertex, visit, weight; //(순서대로) 정점, 방문할 노드, 가중치
};


struct Graph
{
    int V; // 정점의 갯수
    int line; //간선의 갯수

    struct  Node* node; //노드 구조체 포함
};

struct Graph* distanceGraph(int V, int line)
{
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph)); //graph 구조체에 할당

    graph->V = V; //정점의 갯수 V를 graph의 V로 지정
    graph->line = line; //간선의 갯수 line을 graph의 line으로 지정
    graph->node = (struct  Node*) malloc(graph->line * sizeof(struct  Node)); //그래프 구조체 안에 노드 구조체에서 노드의 수 일정

    return graph; //반환
}

void ShortDist(int Distance[], int n) //최단경로 함수
{
    cout << "<vertex의 최단경로>" << endl;
    int i;

    for (i = 0; i < n; ++i) {
        cout << "dist[" << i << "]:  " << Distance[i] << endl; //최단 경로 Distance[i] 출력

    }
}
```

출력

출력 보기 선택(S): 빌드

```
void BellmanFord(struct Graph* graph, int vertex) //벨만포드 함수
{

    int V = graph->V; //graph의 V로 지정

    int line = graph->line; //graph line으로 지정

    int *Distance;
    Distance = (int *)malloc(V * sizeof(int)); //Distance[V] 선언
                                               //그냥 int Distance[V];로 선언할 시 상수가 없다고 에러 뜸

    int i, j;

    for (i = 0; i < V; i++) //i가 노드의 갯수만큼 증가할 때
        Distance[i] = INT_MAX; //거리 값 무한대로 지정


    Distance[vertex] = 0; //노드(정점) 시작점 0으로 설정

    for (i = 1; i <= V - 1; i++) //i가 V-1만큼 증가 할 때
    {
        for (j = 0; j < line; j++) //j가 간선의 갯수만큼 도달할 때
        {
            int w = graph->node[j].vertex; //시작점이 무한대가 아닐 때
            int v = graph->node[j].visit; //다음 노드로 방문
            int weight = graph->node[j].weight;

            if (Distance[w] + weight < Distance[v]) //시작점부터 최단거리에 가중치를 더한 값이
                Distance[v] = Distance[w] + weight;  //도착노드의 가중치보다 작을 때 바뀜
        }
    }

    //음의 가중치로 경로가 무한히 작아지는 경우
    for (i = 0; i < line; i++) //간선의 갯수만큼 증가
    {
        int w = graph->node[i].vertex; //시작점이 무한대가 아닐 때
        int v = graph->node[i].visit; //다음 노드로 방문
        int weight = graph->node[i].weight;


        if (Distance[w] + weight < Distance[v]) //현재 노드에서의 최단거리와 가중치의 합이 계속 작아질 경우
            cout << "음의 싸이클"<<endl; //음의 싸이클 출력
    }

    ShortDist(Distance, V); //최단 경로 함수

    return;
}
```

출력

출력 보기 선택(S):

1>12181748 김소원

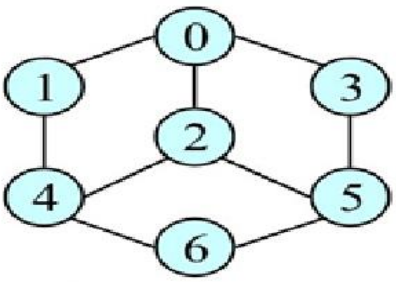
1>C:₩Users₩zzsow₩

```
94  int main() //메인함수
95  {
96      int V, line, F;  //(순서대로) 노드의 갯수, 간선의 갯수, 시작점
97
98      cout << "노드(정점)의 갯수: ";
99      cin >> V; //노드 갯수 입력
100
101     cout << "간선의 갯수: ";
102     cin >> line; //간선 갯수 입력
103
104     cout << "제일 처음 시작하는 노드: ";
105     cin >> F; //처음 시작 노드 입력
106     cout << endl;
107
108     struct Graph* graph = distanceGraph(V, line);   //함수 공간 할당
109
110     int i;
111
112     cout << "시작점, 도착점, 가중점 차례대로 나타내기" << endl;
113     for (i = 0; i<line; i++) { //i가 간선의 갯수만큼 증가할 때
114         cin >> graph->node[i].vertex; //시작점
115         cin >> graph->node[i].visit; //도착점
116         cin >> graph->node[i].weight; //가중점
117     }
118
119     cout << endl;
120
121     BellmanFord(graph, F); ////생성된 그래프와 노드를 벨만포드 함수에 입력
122
123     return 0;
124 }
```

4. 분석 및 결론

1. BFS, DFS 알고리즘

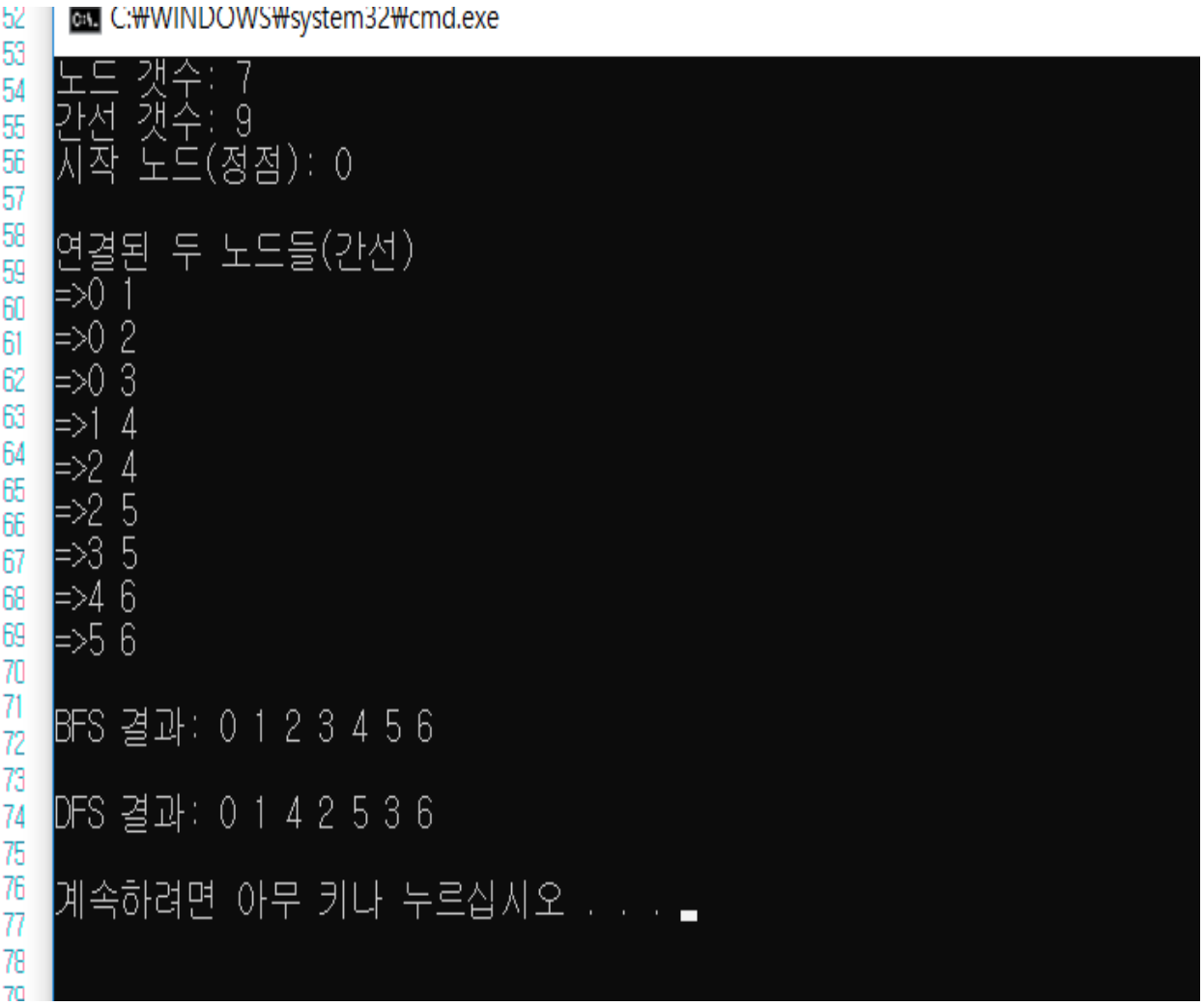
(a) "알고리즘설계_2020_Graph_6월9일_주석.pdf"의 21페이지의 그래프를 대상으로 정상 동작함 보이기



탐색을 위한 그래프 G

위 그래프가 바로 21페이지의 그래프이다.

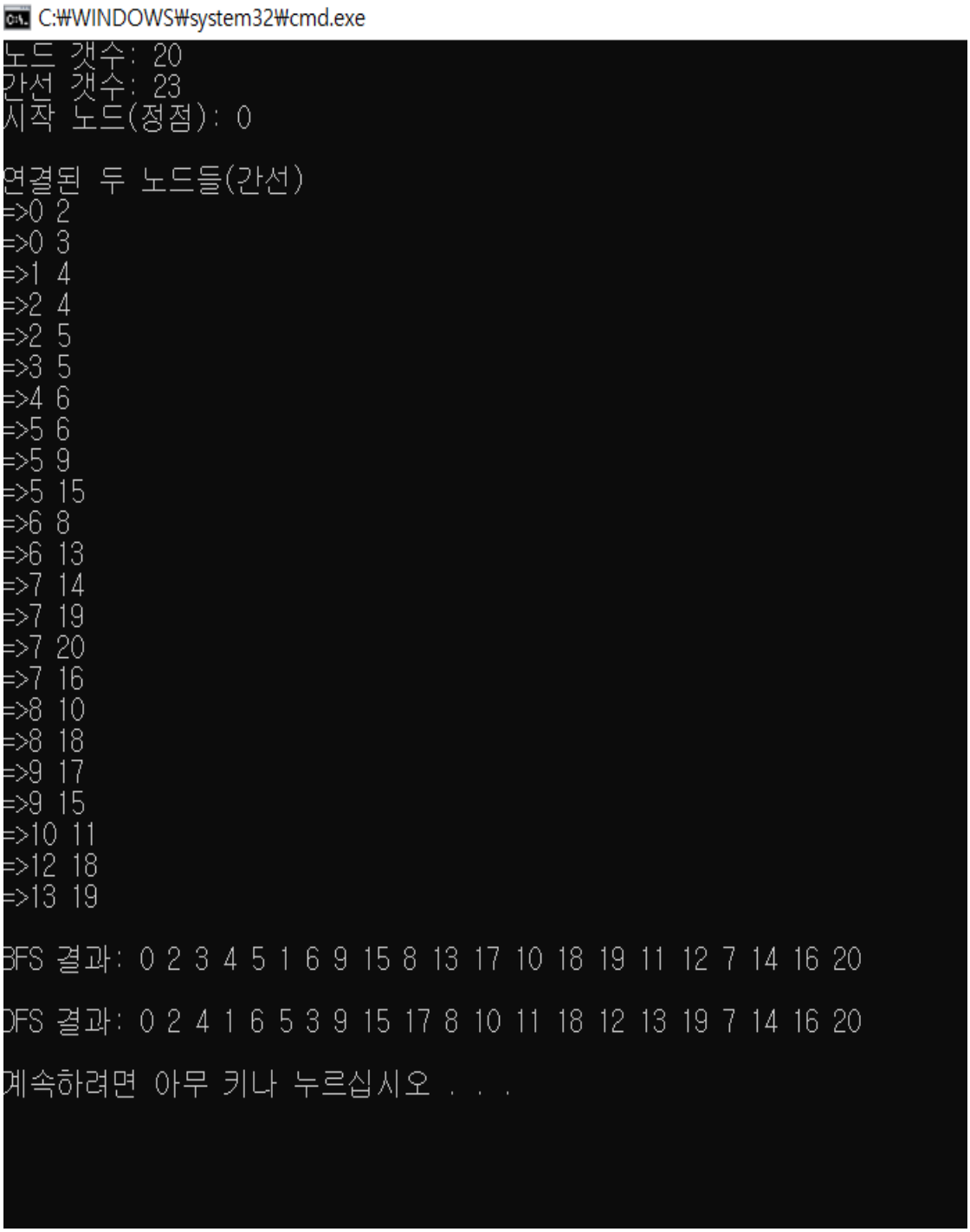
이 그래프를 보면 노드의 개수는 0, 1, 2, 3, 4, 5, 6=> 총 7개이며 간선은 0-1, 0-2, 0-3, 1-4, 2-4, 2-5, 3-5, 4-6, 5-6 총 9개이다. 최종 결과 뿐 아니라 간선들도 보이기 위해 연결된 두 노드들을 입력 받았다. 0을 시작점으로 정하고 BFS를 직접 풀어보면 가장 가까운 노드인 1,2,3으로 먼저 향하고 그 다음 가까운 4, 5 마지막으로 6을 방문하므로 BFS의 결과는 0 1 2 3 4 5 6이 나올 것이다. DFS로 풀어보면 0에서 시작하여 1로 간 후 1과 연결된 4로 갈 것이다. 4는 2와 6이 연결되어 있는데 더 작은 숫자인 2로 향하고 그 다음 5를 방문한 후 3을 방문할 것이다. 3까지 방문하면 더 이상 방문할 노드가 없으므로 다시 최상단 노드를 빼 5까지 다시 돌아간 후 마지막으로 6을 방문할 것이다. 따라 결과는 0 1 4 2 5 3 6이 나올 것이다.



그 결과 BFS 결과는 0 1 2 3 4 5 6이 나왔으며 DFS 결과는 0 1 4 2 5 3 6이 나왔으며 내가 직접 풀어본 값과 같은 결과가 나왔음을 알 수 있다.

(b) 노드의 수가 20개 이상인 그래프를 임의로 정의하고 정상 동작함을 보여라

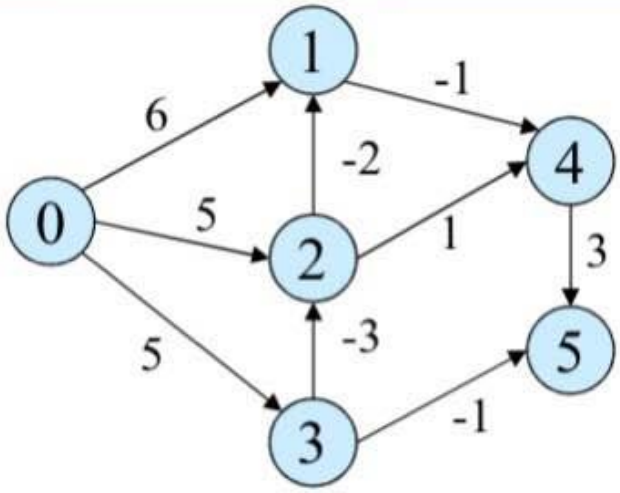
나는 노드의 개수를 0에서 19까지 총 20개로 정하고 간선의 개수를 23개라고 임의로 정의하였다. 시작 노드를 0으로 설정한 후 연결된 두 노드들 총 23개를 아래 결과값과 같이 나타내 주었다.



그 결과 BFS는 0 2 3 4 5 1 6 9 15 8 13 17 10 18 19 11 12 7 14 16 20이 나왔으며 DFS 결과는 0 2 4 1 6 5 3 9 15 17 8 10 11 18 12 13 19 7 14 16 20이 나왔다. 이는 위에서 설명한 것과 동일한 방법으로 나온 값임을 알 수 있었다.

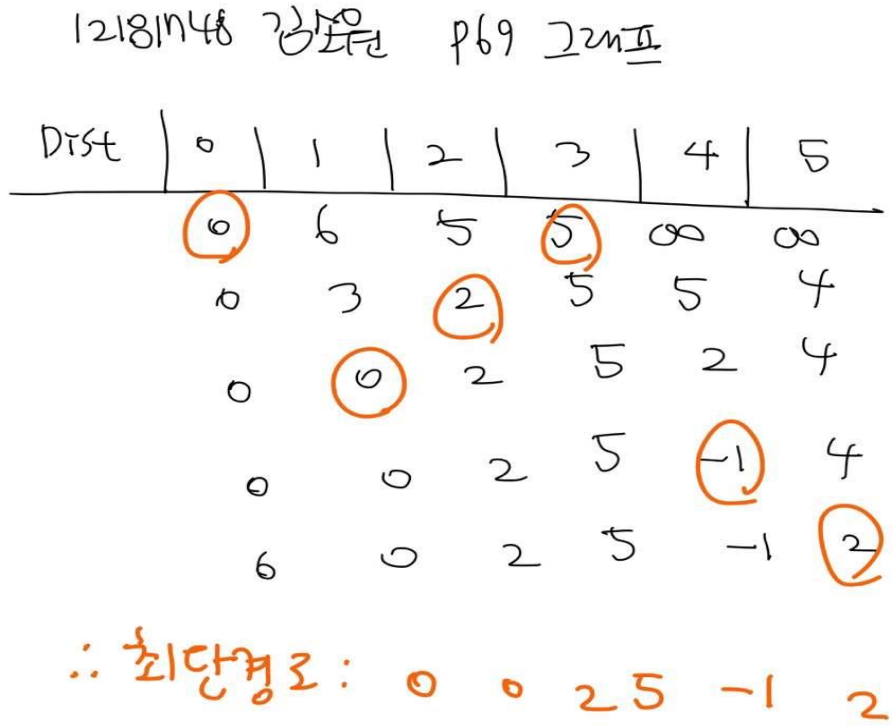
2. Bellman Ford 알고리즘

(b) "알고리즘설계_2020_Graph_6월9일_주석.pdf"의 69,72페이지의 그래프를 대상으로 각각 정상 동작함 보이기



(a) 방향 그래프(시작점 0)

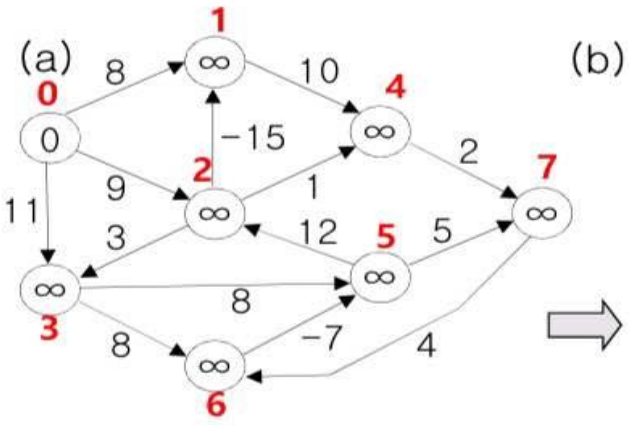
위 그래프가 69페이지 그래프이다. 먼저 69페이지 그래프가 정상 동작하는지 보기 위해서 살펴보면 이 그래프의 노드의 개수는 0, 1, 2, 3, 4, 5 => 총 6개이며 간선의 개수는 총 9개이다.



위 그래프의 최단 경로를 구하기 위한 설명을 그림으로 한번 나타내 보았는데 젤 윗줄이 바로

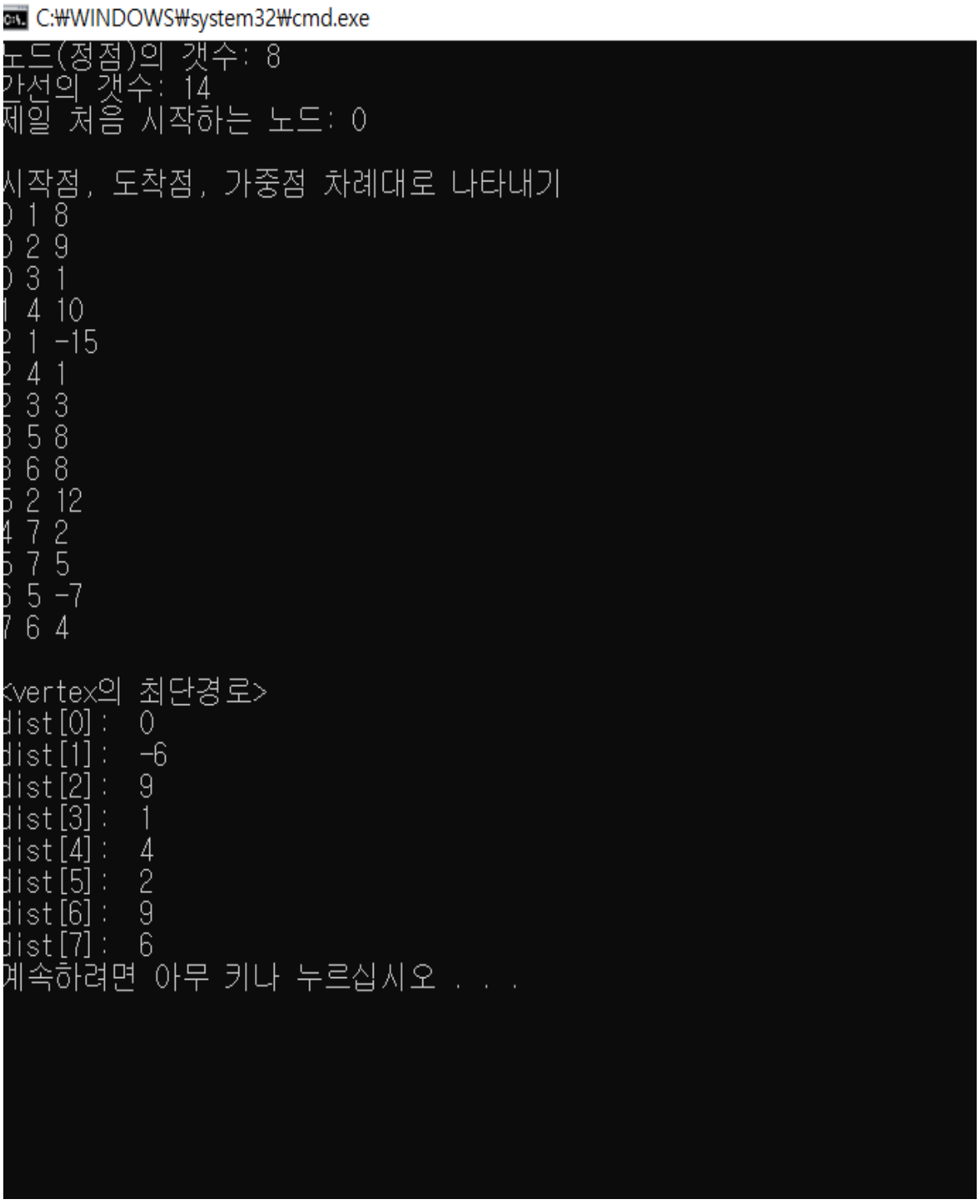
시작점 0에서 한번 이동했을 때의 값들이다. 0에서 4,5로 가는 값이 무한대인 이유는 한번에 가는 값은 없기 때문이다. 이렇게 값들을 구한 후 가장 작은 값을 찾으면 그 값이 바로 최단 경로가 된다. 따라 최단 경로는 0 0 2 5 -1 2가 나오며 아래와 같이 동일한 결과값이 나오는 것을 알 수 있었다.

```
C:₩WINDOWS₩system32₩cmd.exe
노드(정점)의 갯수: 6
간선의 갯수: 9
제일 처음 시작하는 노드: 0

시작점, 도착점, 가중점 차례대로 나타내기
0 1 6
2 1 -2
0 2 5
0 3 5
1 4 -1
2 4 1
3 5 -1
4 5 3
3 2 -3

<vertex의 최단경로>
dist[0]:  0
dist[1]:  0
dist[2]:  2
dist[3]:  5
dist[4]:  -1
dist[5]:  2
계속하려면 아무 키나 누르십시오 . . .
```



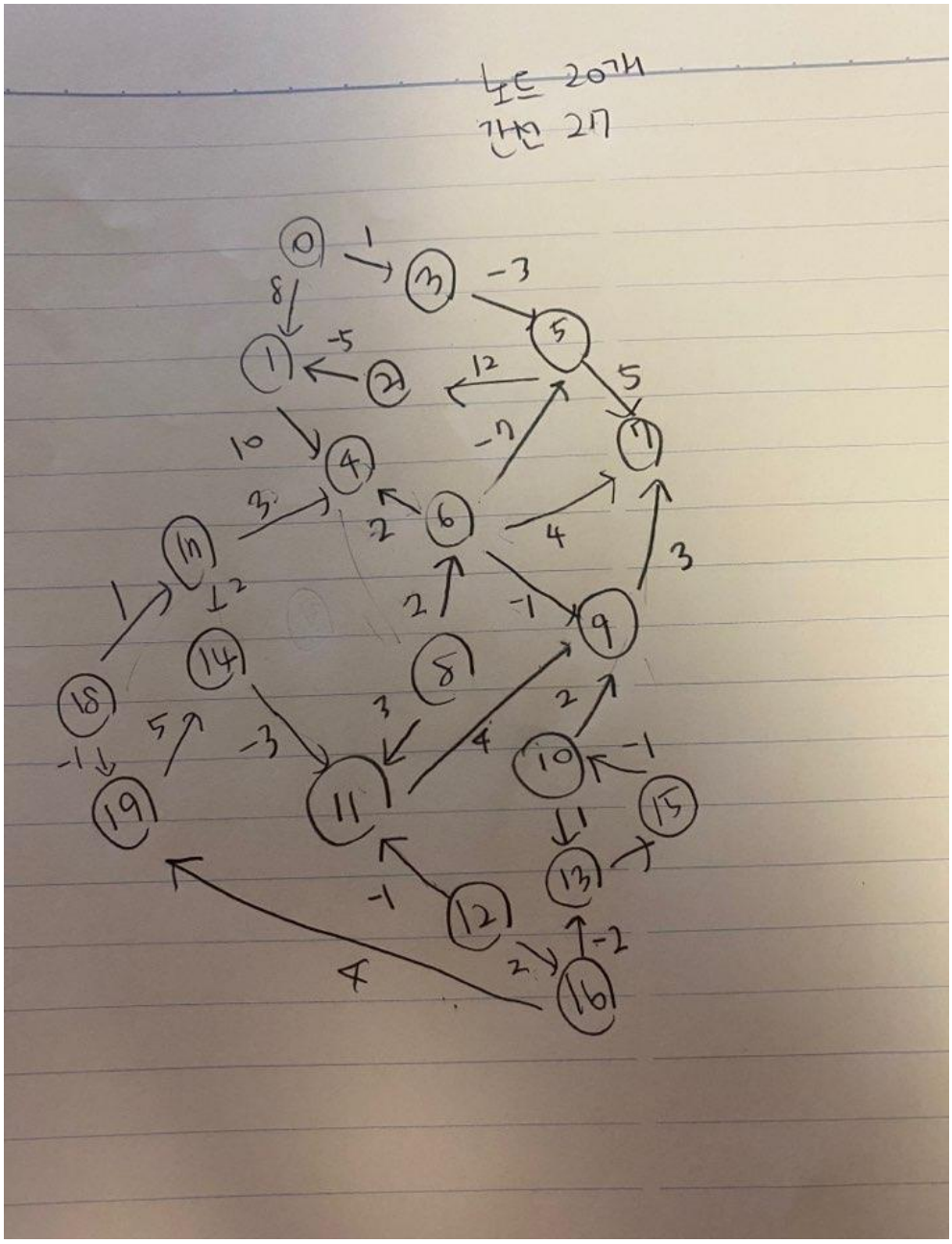
위 그래프는 72페이지의 그래프로 노드의 개수는 0부터 7까지 총 8개이며 간선의 개수는 14

개이다. 시작점 역시 위와 같이 0이 제일 처음 시작하는 노드이며 시작점, 도착점, 가중점을 차례대로 나타내보면 0 1 8/ 0 2 9/ 0 3 1/ 1 4 10/ 2 1 -15/ 2 4 1/ 2 3 3/ 3 5 8/ 3 6 8/ 5 2 12/ 4 7 2/ 5 7 5/ 6 5 -7/ 7 6 4=> 총 14개이다. 여기서 0 1 8은 노드 0에서 노드1로 가는 가중치 값이 8이란 뜻이다. 이렇게 이 그래프의 최단경로를 벨만포드 알고리즘을 이용하여 구하면 최단경로는 0 -6 9 1 4 2 9 6이 나오는 것을 알 수 있다.



(c) 노드의 수가 20개 이상인 그래프를 임의로 정의하고 정상 동작함을 보여라.

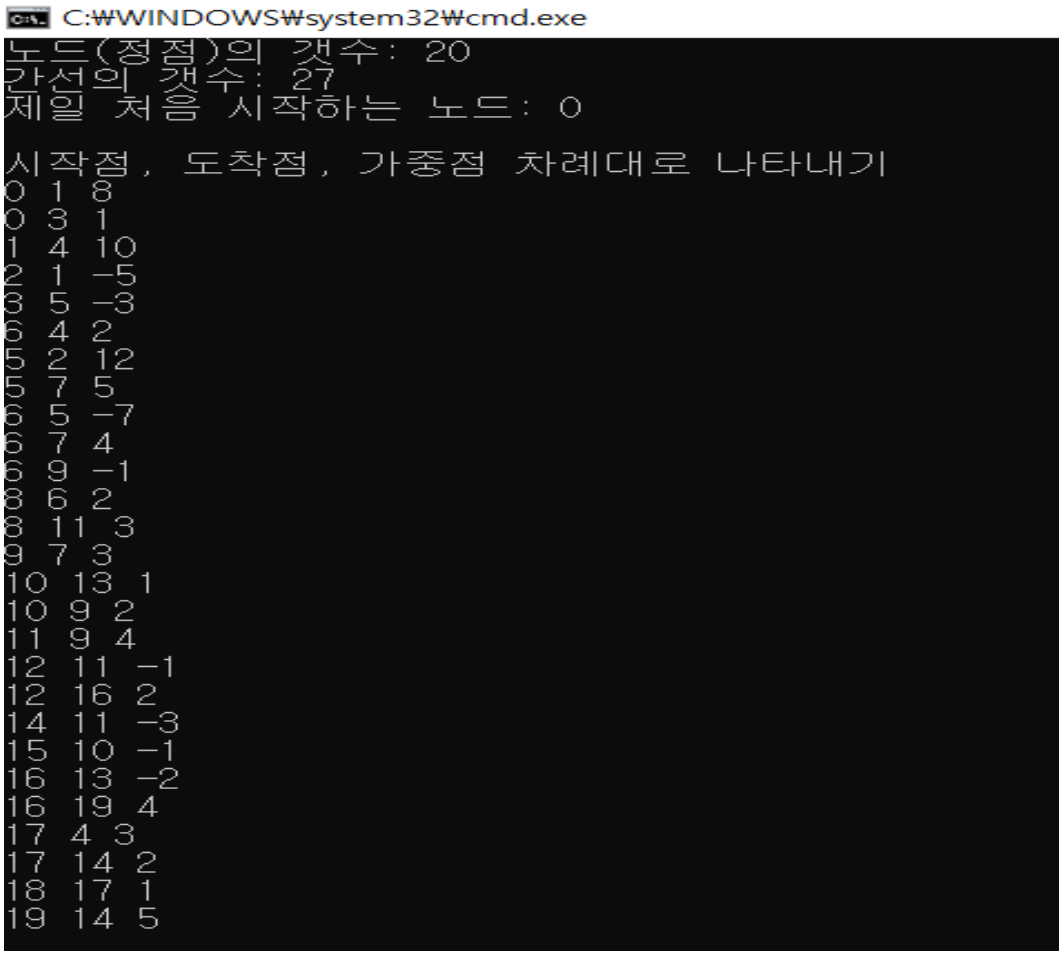
마지막으로 노드가 20개인 그래프를 먼저 임의로 그려보았다.



노드의 개수는 0부터 19까지 총 20개이며 간선의 개수는 27개로 설정하였다.

제일 처음 시작하는 노드를 0이라 설정하였으며 시작점, 도착점, 가중치는 0 1 8/ 0 3 1/ 1 4 10 /2 1 -5/ 3 5 -3/ 6 4 2/ 5 2 12 /5 7 5 / 6 5 -7/ 6 7 4/ 6 9 -1/ 8 6 2/ 8 11 3/ 9 7 3/ 10 13 1/ 10 9 2/ 11 9 4/ 12 11 -1/ 12 16 2/ 14 11 -3/ 15 10 -1 / 16 13 -2 /16 19 4 /17 4 3 /17 14 2 /18 17 1 /19 14 5 => 27개이다

결과를 보면 아래와 같다.



이때 vertex 최단 경로를 보면 -2147483647과 같은 값은 무한히 크거나 작은 값들이 있는데 이는 int가 나타낼 수 있는 최소/최대 값으로 음의 싸이클이 적용되었기 때문에 생긴 것이라 할 수 있다.

BFS는 최단 경로가 존재하면 깊이에 상관없이 답을 찾을 수 있으며 DFS는 찾으려는 노드가 깊은 단계일 경우 BFS 보다 빠르다는 것을 알게 되었다. 이와 같이 이번 과제를 통해 BFS와 DFS의 각 장단점을 잘 알게 되었으며 BFS는 큐를 사용하며 여러 제약이 있을 경우는 DFS가 더 좋은 것과 같이 이들의 탐색 차이도 자세히 알게 되었다. Bellman ford 알고리즘은 다익스트라 알고리즘과 비슷하지만 음의 가중치도 적용된다는 차이점이 있다는 것을 인지하고 음의 순환이 반복되지 않게 주의할 필요가 있다는 것을 코드를 짜면서 더욱 느끼게 되었다. 나는 요번 과제를 통해 습득한 내용을 통해 나중에 코드를 짤 때 어떤 알고리즘이 더 효율적인지 빠르게 판단하여 더 나은 코드를 짜도록 할 것이다.

5. 참고문헌 및 참고자료

- 유투브 컴퓨터 알고리즘 기초 17 강 벨만-포드 알고리즘| T 아카데미

- 유투브 깊이 우선 탐색(Depth First Search) [실전 알고리즘 강좌(Algorithm Programming Tutorial) #17]