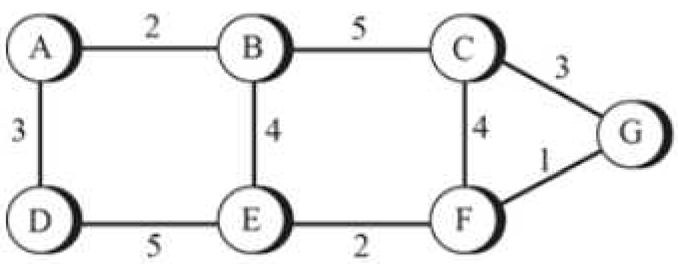
컴퓨터공학과 20154215 구명회 Computer Networks Homework #2



	A	B	C	D	E	F	G
A	0	2	0	3	0	0	0
B	2	0	5	0	4	0	0
C	0	5	0	0	0	4	3
D	3	0	0	0	5	0	0
E	0	4	0	5	0	2	0
F	0	0	4	0	2	0	1
G	0	0	3	0	0	1	0

<Example network graph and its adjacency matrix >

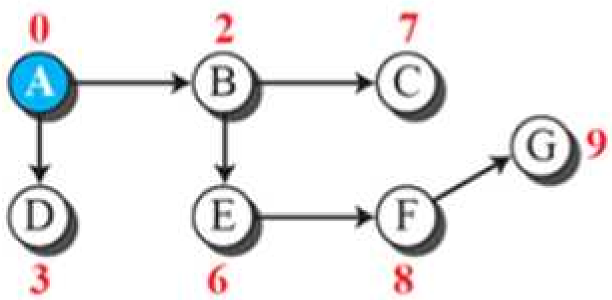
%%

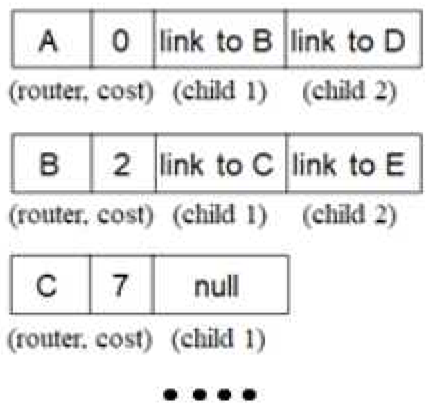
% Program to find the least-cost tree for unicast routing %

%%

\> Input the source router: A

<Example keyboard input>





<Example output tree and its linked list>

%%

% Program to find the least-cost tree for unicast routing %

%%

\> Input the source router: A

\> Least-cost tree rooted at A: (A, 0, B, D), (B, 2, C, E), (C, 7, null), (D, 3, null), (E, 6, F), (F, 8, G), (G, 9, null)

\> Input the source router: B

\> Least-cost tree rooted at B: (A, 2, D), (B, 0, A, C, E), (C, 5, null), • • • •

<Example output>

\>>> 1. 코드

\>>> 2. 프로그램 실행 사진

\>>> 3. 프로그램 실행 순서

\>>> 4. 과제를 마친 소감

>>> 1. 코드는 C언어로 작성했으며, 추가로 C파일 첨부하겠습니다

```c

////////////////////////////////// 매크로 상수 & 전역 변수 & 헤더 파일 ////////////////////////////////////////

#define _CRT_SECURE_NO_WARNINGS
#define NO_CONNECTION 19961204                  // 연결이 없는 경우 (큰 정수)
#define PEAK_NUMBER 6                           // 정점의 수

int DUPLICATE_ROUTE[PEAK_NUMBER][PEAK_NUMBER];  // 중복 경로 저장
int Matrix_path[PEAK_NUMBER][PEAK_NUMBER];      // 최단거리 정점까지 거치는 노드들을 저장
int COST[PEAK_NUMBER][PEAK_NUMBER];             // 입력받을 행렬
int DISTANCE[PEAK_NUMBER];                      // 시작 정점으로부터의 최단 경로 거리
int FOUND[PEAK_NUMBER];                         // 방문한 정점 표시

#include <stdio.h>
#include <stdlib.h>
////////////////////////////////// 매크로 상수 & 전역 변수 & 헤더 파일 ////////////////////////////////////////

// (10)
void
Print_Route(int start) {
    ////////////////////////////////// 출력 ////////////////////////////////
    // 저장된 경로들을 (A,0,B,C) (B,2,C,E) ... 와 같이 출력합니다
    // 1. 시작 정점과 시작 정점에서의 각각의 거리를 출력합니다.
    // 2. DUPLICATE_ROUTE은 최종적으로 한 정점에서 중복된 경로가 삭제되어 있고,
    // 경로가 0번째 인덱스부터 저장되어 있으므로,
    // 0번째 인덱스부터 출력을 하되, 인덱스가 연결되지 않은 경우가 나오면 출력하지 않습니다.

    printf(" > Least-cost tree rooted at %c : ", start + 65);
    for (int i = 0; i < PEAK_NUMBER; i++) {
        printf("(%c, %d", i + 65, DISTANCE[i]);

        for (int j = 0; j < PEAK_NUMBER; j++) {
            if (DUPLICATE_ROUTE[i][0] == NO_CONNECTION) {
                printf(", NULL");
                break;
            }
            if (DUPLICATE_ROUTE[i][j] != NO_CONNECTION)
                printf(", %c", DUPLICATE_ROUTE[i][j] + 65);
        }
        printf(")");
        if (i != PEAK_NUMBER - 1)
            printf(", ");
    }

    printf("₩n₩n");
    ////////////////////////////////출력 끝////////////////////////////
}

// (9)
void
Delete_Duplicate_Route() {
    ////////////////////// 중복 경로 삭제 ////////////////////////
    // 1. DUPLICATE_ROUTE의 첫번째 열의 원소들을 나머지 열들의 원소들과 비교해서
    // 같은 경로가 있다면 나머지 열에 저장된 경로를 연결이 없음으로 바꿔줍니다.
    // 2. 첫번째 열에서 중복되는 경로를 NO_CONNECTION 제외하고 찾습니다.
    // DUPLICATE_ROUTE에는 중복되는 경로의 열이 저장되어 있습니다.
    // Matrix_path에는 최단거리 경로 거리가 저장되어 있습니다.
    // 같은 열에서 NO_CONNECTION를 제외하고 중복된 경로를 찾으면
    // 그 열의 다음 원소를 그 앞의 원소로 옮겨줍니다. (2 -> 1, 3 -> 2, 4 -> 3, ...)

    // 중복 경로 삭제하기위한 변수,,,
    int delete_route = 0;

    for (int one = 0; one < PEAK_NUMBER; one++)
        for (int two = 0; two < PEAK_NUMBER; two++) {
            for (int thr = 1; thr < PEAK_NUMBER; thr++) {
                for (int fou = 0; fou < PEAK_NUMBER; fou++) {
                    if (((two == fou) && (one == thr)))
                        break;
                    if (DUPLICATE_ROUTE[two][one] == DUPLICATE_ROUTE[fou][thr]) {
                        DUPLICATE_ROUTE[fou][thr] = NO_CONNECTION;
                    }
                }
            }
        }

```

```
77
78      for (int one = 0; one < PEAK_NUMBER; one++)
79          for (int two = 0; two < PEAK_NUMBER; two++)
80              for (int thr = two + 1; thr < PEAK_NUMBER; thr++)
81                  if ((DUPLICATE_ROUTE[two][one] == DUPLICATE_ROUTE[thr][0])) {
82                      if (DUPLICATE_ROUTE[two][one] != NO_CONNECTION) {
83                          delete_route = DUPLICATE_ROUTE[two][one];
84                          if (Matrix_path[two][delete_route] < Matrix_path[thr][delete_route])
85                              for (int fou = one; fou < PEAK_NUMBER - 1; fou++)
86                                  DUPLICATE_ROUTE[thr][fou] = DUPLICATE_ROUTE[thr][fou + 1];
87                          else
88                              for (int fiv = one; fiv < PEAK_NUMBER - 1; fiv++)
89                                  DUPLICATE_ROUTE[two][fiv] = DUPLICATE_ROUTE[two][fiv + 1];
90                      }
91                  }
92      ////////////////////// 중복 경로 삭제 끝 ////////////////////////
93  }
94
95  // (8)
96  int
97  Find_Matrix_Vertex( ) {
98      ////////////////////// 최단거리에 있는 정점 찾기 ////////////////////
99      // 1. 현재 DISTANCE[]에서 가장 작은 가중치 값이 위치하고 있는 배열의 인덱스를 찾아 반환합니다.
100     // 2. 최소값 min을 찾아 방문하지 않은 정점에 대해 DISTANCE[] 값을 비교합니다.
101     // 방문한 적 없는 정점이고 현재까지의 최소값보다 작으면 최소값을 갱신하고
102     // 최소값이 등장했으므로 mp에 정점의 인덱스를 저장합니다
103     // 3. 최소값을 반환합니다.
104
105     int min = NO_CONNECTION;
106     int mp = -1;
107
108     for (int minima = 0; minima < PEAK_NUMBER; minima++)
109         if (DISTANCE[minima] < min && !FOUND[minima]) {
110             min = DISTANCE[minima];
111             mp = minima;
112         }
113
114     return mp;
115     ////////////////////// 최단거리에 있는 정점 찾기 끝 ////////////////////
116 }
117
118 // (7)
119 void
120 Route_Storage(int start) {
121     ////////////////////// 중복 경로 저장하기 ////////////////////
122     // 1. 시작하는 정점은 방문했음(TRUE)으로 표시합니다.
123     // 2. 시작 지점을 지정해줍니다.
124     // 3. 중복 경로를 저장합니다.
125     // 마지막에 경로를 출력할 때 배열을 사용해서 출력하므로,
126     // 경로를 0부터 차례대로 저장해야합니다.
127     // 아직 방문하지 않은 정점들 중에서
128     // txt 파일로부터 받은 Matrix가 저장된 COST의 값들을 비교해서 연결된 부분을 찾고,
129     // 현재 그 정점까지의 거리 + 다음 정점까지 거리가
130     // 기존 정점까지 거리보다 같거나 가까우면 DUPLICATE_ROUTE에 인덱스를 저장합니다.
131     // DUPLICATE_ROUTE는 현재 노드에서 갈 수 있는 정점을 출력할 때 사용합니다.
132     // 4. Find_Matrix_Vertex( ) 함수를 호출해 최소값이 있는 인덱스를 찾고 그 인덱스에 방문표시를 합니다.
133     // 5. 아직 방문하지 않은 정점들 중에서
134     // 현재 그 정점까지의 거리 + 다음 정점까지 거리가
135     // 기존 정점까지 거리보다 가까우면 DISTANCE와 Matrix_path를 갱신합니다.
136
137     FOUND[start] = 1;
138     DISTANCE[start] = 0;
139
140     // 입력을 A로 했을때 >>> u = A 입니다.
141     //       B          >>> u = B 입니다.
142     int Display = start;
143
144     // 총 정점 개수에서 시작 정점 하나를 뺀만큼 반복해야 합니다.
145     for (int i = 0; i < PEAK_NUMBER - 1; i++) {
146
147         // 중복 경로를 저장합니다.
148         int val = 0;
149
150         for (int save = 0; save < PEAK_NUMBER; save++)
151             if (!FOUND[save] && COST[Display][save] != NO_CONNECTION)
152                 if (DISTANCE[Display] + COST[Display][save] <= DISTANCE[save]) {
153                     DUPLICATE_ROUTE[Display][val] = save;
154                     val++;
155                 }
156
```

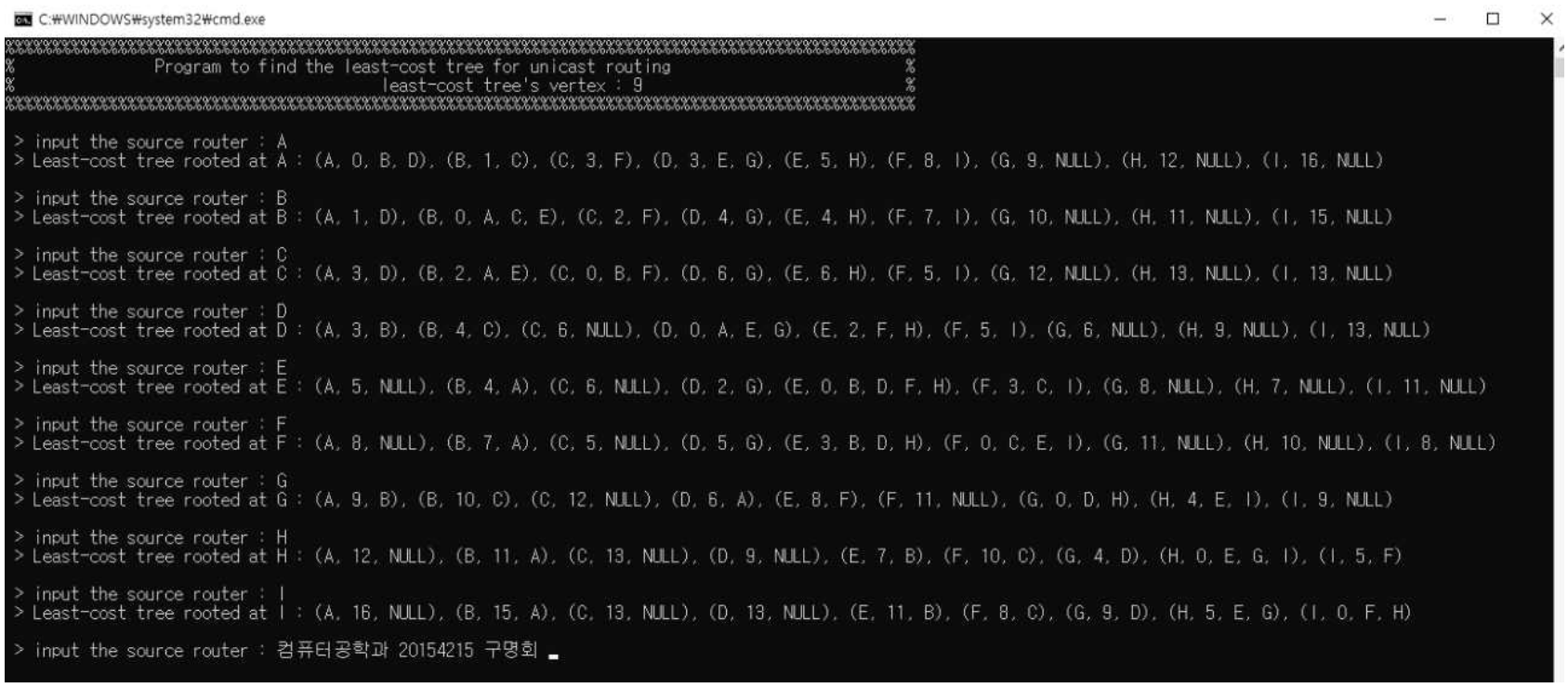
```
157             // 방문표시를 합니다.
158             Display = Find_Matrix_Vertex();
159             FOUND[Display] = 1;
160
161             // 정보를 갱신합니다.
162             for (int Re = 0; Re < PEAK_NUMBER; Re++)
163                 if (!FOUND[Re])
164                     if (DISTANCE[Display] + COST[Display][Re] < DISTANCE[Re])
165                     {
166                         DISTANCE[Re] = DISTANCE[Display] + COST[Display][Re];
167                         Matrix_path[Display][Re] = DISTANCE[Re];
168                     }
169         }
170         ////////////////////// 중복 경로 저장하기 끝 ////////////////////
171     }
172
173     // (6)
174     void
175     Find_Shortest_Path(int start) {
176         ////////////////////  최단 경로 거리 구하기 ////////////////////
177         // 1. DISTANCE, FOUND, Matrix_path를 각각 초기화 합니다.
178         // DISTANCE는 시작 정점으로부터의 최단 경로 거리이므로 현재 입력받은 정점의 행
179         // 즉 A를 입력받았을 때 A에서의 최단 경로 거리는 [ 0, 2, 0, 3, 0, 0, 0 ]가 됩니다.
180         // FOUND는 방문한 정점을 표시하므로, 현재 방문한 정점이 없으므로 FALSE입니다.
181         // Matrix_path는 최단거리 정점까지 거치는 노드들을 저장하므로 현재 입력받은 정점의
182         // 인덱스를 0번째 열에 저장합니다.
183         // Matrix_path는 DUPLICATE_ROUTE와 비교하며 중복 경로를 찾습니다.
184         // 2. Route_Storage() 함수를 호출하여 중복 경로를 저장합니다.
185         // 3. Delete_Duplicate_Route() 함수를 호출하여 중복 경로를 삭제합니다.
186         // 4. Print_Route() 호출하여 시작 정점부터 마지막 정점까지 최단 경로 출력합니다.
187
188         for (int i = 0; i < PEAK_NUMBER; i++) {
189             DISTANCE[i] = COST[start][i];
190             FOUND[i] = 0;
191             Matrix_path[i][0] = start;
192         }
193
194         Route_Storage(start);
195         Delete_Duplicate_Route();
196         Print_Route(start);
197         ////////////////////  최단 경로 거리 구하기 끝 ////////////////////
198     }
199
200     // (5)
201     void
202     Array_init(int arr[][PEAK_NUMBER]) {
203         ////////////////////// 초기화 ////////////////////////////
204         // 모든 Matrix의 값을 연결되지 않음으로 바꿔줍니다.
205         // 초기화하는 Matrix들은
206         // 최단거리를 구할 때, 중복된 경로를 삭제할 때 사용합니다.
207
208         for (int i = 0; i < PEAK_NUMBER; i++)
209             for (int j = 0; j < PEAK_NUMBER; j++)
210                 arr[i][j] = NO_CONNECTION;
211         ////////////////////// 초기화 끝 ////////////////////////////
212     }
213
214     // (4)
215     void
216     InputOutput_Running_Test() {
217         ////////////////////// 프로그램 시작 ////////////////////////////
218         // 1. 시작 정점 입력받습니다.
219         // 시작 정점을 char로 입력받게 되면, enter까지 버퍼에 저장되기 때문에
220         // getchar()를 사용해 ₩n을 삭제했습니다.
221         // 2. 먼저 Matrix_path와, DUPLICATE_ROUTE를 Array_init() 함수를 호출하여 초기화합니다.
222         // Matrix_path와 DUPLICATE_ROUTE는 각각 전역변수로 선언되어 있습니다.
223         // 3. Find_Shortest_Path() 함수를 호출하여 최단 경로를 구합니다.
224
225         char start, START;
226
227         while (1) {
228             printf(" > input the source router : ");
229             scanf_s("%c", &start, 1);
230             START = getchar(start != '₩n');
231
232             Array_init(Matrix_path);        Array_init(DUPLICATE_ROUTE);
233             Find_Shortest_Path(start - 65);
234         }
235         ////////////////////// 프로그램 끝 ////////////////////////////
236     }
```

```c

// (3)
void
Read_Cost(int Matrix[][PEAK_NUMBER], FILE* file) {
    ///////////////////////// txt파일 변경 //////////////////////////
    // txt파일을 읽어왔습니다.
    // 현재 과제에서 읽어온 Matrix는 다음과 같습니다.
    //  0, 2, 0, 3, 0, 0, 0
    //  2, 0, 5, 0, 4, 0, 0
    //  0, 5, 0, 0, 0, 4, 3
    //  3, 0, 0, 0, 5, 0, 0
    //  0, 4, 0, 5, 0, 2, 0
    //  0, 0, 4, 0, 2, 0, 1
    //  0, 0, 3, 0, 0, 1, 0
    // 값이 0인 부분은 정점끼리 서로 연결되지 않았습니다.
    // 계산을 편하게 하기 위해 연결되지 않은 성분은
    // 매크로 상수인 NO_CONNECTION로 바꿔줍니다.
    // 매크로 상수 NO_CONNECTION는 큰 정수값을 가지고 있습니다.

    for (int i = 0; i < PEAK_NUMBER; i++)
        for (int j = 0; j < PEAK_NUMBER; j++) {
            int r = fscanf(file, "%d ,", Matrix[i] + j);

            if (r < 1) {
                fprintf(stderr, "자료가 부족합니다.₩n");
                exit(-1);
            }
            // 연결되지않은 노드의 값 변경
            if (Matrix[i][j] == 0)
                Matrix[i][j] = NO_CONNECTION;
        }
    ///////////////////////// txt파일 변경 끝 //////////////////////////
}

// (2)
void
File_InputOutput() {
    ////////////////////// 파일 입출력 /////////////////////////
    // txt파일을 읽어옵니다.

    FILE* fp = fopen("./Matrix_txt_6x6.txt", "r");

    if (fp == NULL) {
        puts("file open failed!!");     return -1;
    }

    Read_Cost(COST, fp);     fclose(fp);
    ////////////////////// 파일 입출력 끝 /////////////////////////
}

// (1)
void
Compile_Message() {
    ////////////////////// 메세지를 출력합니다 /////////////////////////
    // 메세지와 현재 정점의 갯수를 출력하며, 정점의 개수는 매크로 상수로 정의됩니다.

    printf("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    printf("%%      Program to find the least-cost tree for unicast routing            %%₩n");
    printf("%%                  least-cost tree's vertex : %d               %%₩n", PEAK_NUMBER);
    printf("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ////////////////////// 메세지를 출력합니다 끝 /////////////////////////
}

// START
int
main() {
    ////////////////////// main 함수 /////////////////////////
    // 1. message를 출력합니다.
    // 2. txt 파일을 읽어옵니다.
    // 3. 읽어온 txt파일로 다익스트라 알고리즘을 사용하여 각각 정점에서의 경로를 구합니다.

    Compile_Message();
    File_InputOutput();
    InputOutput_Running_Test();
    ////////////////////// main 함수 끝 /////////////////////////
}
```

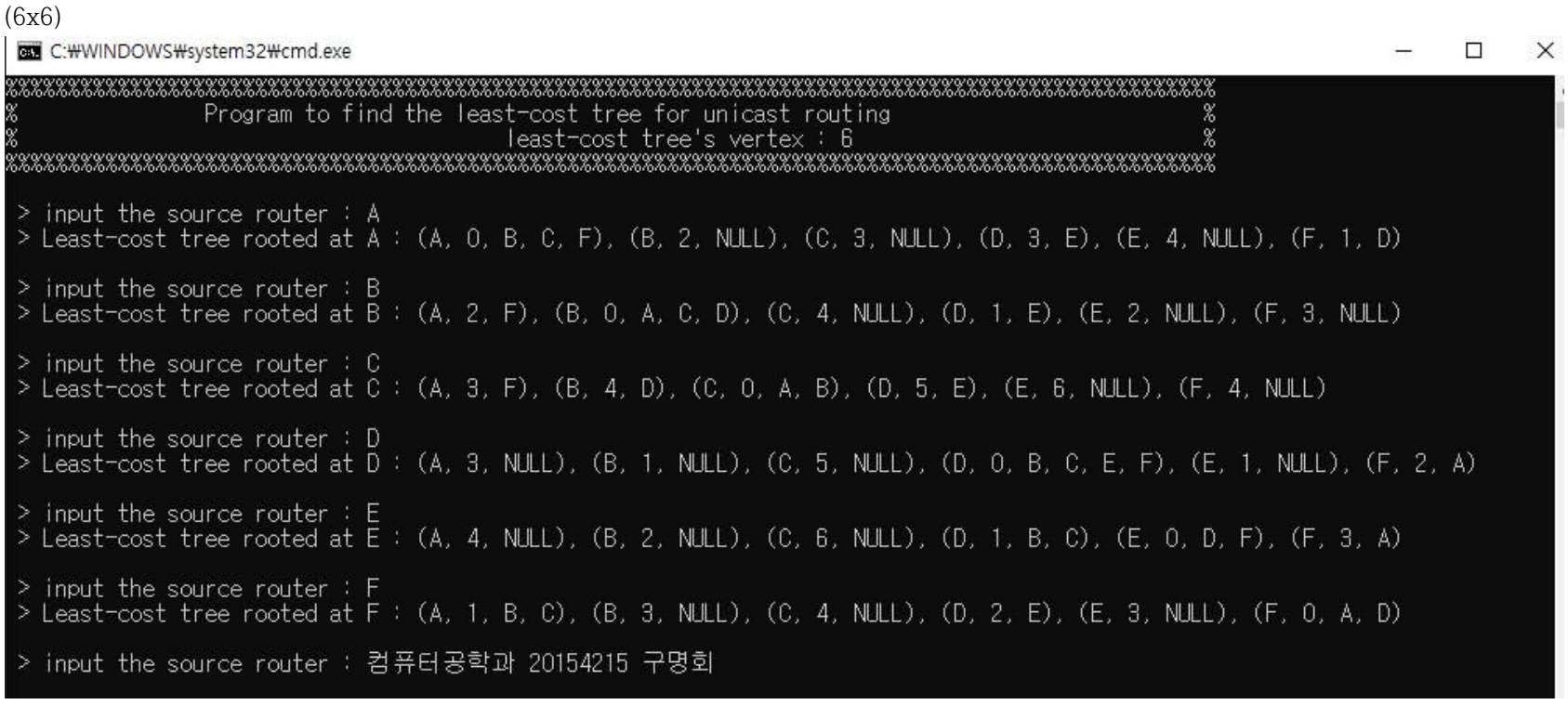
>>> 2. 프로그램 실행 사진, 추가로 각 경로에 대한 그래프 그림, txt파일 첨부하겠습니다

다른 txt파일으로 프로그램을 실행시켜도 정상적으로 작동하는 것을 확인했습니다.

(9x9)



(6x6)



>>> 3. 프로그램 실행 순서

우선 프로그램은 아래의 11개 함수로 구성됩니다.

00) main() 함수

01) Compile_Message() 함수 - 메시지와 현재 정점의 갯수를 출력

02) File_InputOutput() 함수 - txt 파일 읽기

03) Read_Cost() 함수 - 읽어온 txt Matrix 변경

04) InputOutput_Running_Test() 함수 - 프로그램 시작

05) Array_init() 함수 - 초기화

06) Find_Shortest_Path() 함수 - 최단 경로 거리 찾기

07) Route_Storage() 함수 - 중복 경로를 저장

08) Find_Matrix_Vertex() 함수 - 최단거리에 있는 정점 찾기

09) Delete_Duplicate_Route() 함수 - 중복 경로 삭제

10) Print_Route() 함수 - 경로 출력

프로그램의 실행 순서에 대한 함수 설명은 각각 주석으로 작성하였습니다.

>>> 4. 과제를 마친 소감

저는 이번 과제에서, 최소 비용 신장 트리를 구성하는데 사용하는 알고리즘들 중, 다익스트라 알고리즘을 사용하여 구현하였습니다.
특별히 과제하면서 힘들었던점은, 다익스트라 알고리즘과는 별개로 기초적인 프로그램 짜는것부터 기억이 잘 안나서 처음부터 하나하나 찾아보면서 생각보다 시간이 오래걸렸던 부분이었습니다. 사실 처음에 과제를 받았을 때 링크드 리스트로 구현해볼까 싶었지만 마찬가지로 공부가 부족해서... 링크드 리스트는 다음에 조금 더 공부해서 구현해보도록 하겠습니다.

추가적으로, 마지막 날에서야 다른 txt파일(그래프)을 넣어봤는데 그제서야 중복되는 경로를 삭제하는 부분이 잘못된 것을 발견해서 눈물을 머금고 코드 추가했습니다... 때문에, 앞으로는 이런일이 없도록 과제를 제출하기 전에 항상 검토하는 습관을 가져야겠습니다.