자료구조 Project Self-Evaluation

**2012210112** 유현우

|  |  |  |  |
| --- | --- | --- | --- |
| **번호** | **항목** | **만점** | **Self-Evaluation** |
| 1 | Submit a github account | 10 | **10** |
| 2 | Commit source code displaying menu | 10 | **10** |
| 3 | Commit the first draft of manual | 10 | **10** |
| 4 | Read data files | 20 | **20** |
| 5 | Statistics | 20 | **20** |
| 6 | Top 5 most tweeted words | 10 | **9** |
| 7 | Top 5 most tweeted users | 5 | **5** |
| 8 | Find all users who mentioned a word | 10 | **9** |
| 9 | Find all users who are friend of the above user | 5 | **5** |
| 10 | Top 5 strongly connected components | 10 | **10** |
| 11 | Find shortest path from a user (id) | 10 | **10** |

**목차**

내용

[**1.** **Read data files** 3](#_Toc453679062)

[**1.1.** **샘플 User 데이터 생성** 3](#_Toc453679063)

[**1.2.** **샘플 Friend 데이터 생성** 4](#_Toc453679064)

[**1.3.** **샘플 Tweet 데이터 생성** 5](#_Toc453679065)

[**2.** **Statistics** 6](#_Toc453679066)

[**3.** **Top 5 most tweeted words** 7](#_Toc453679067)

[**4.** **Top 5 most tweeted users** 8](#_Toc453679068)

[**5.** **Find all users who mentioned a word** 9](#_Toc453679069)

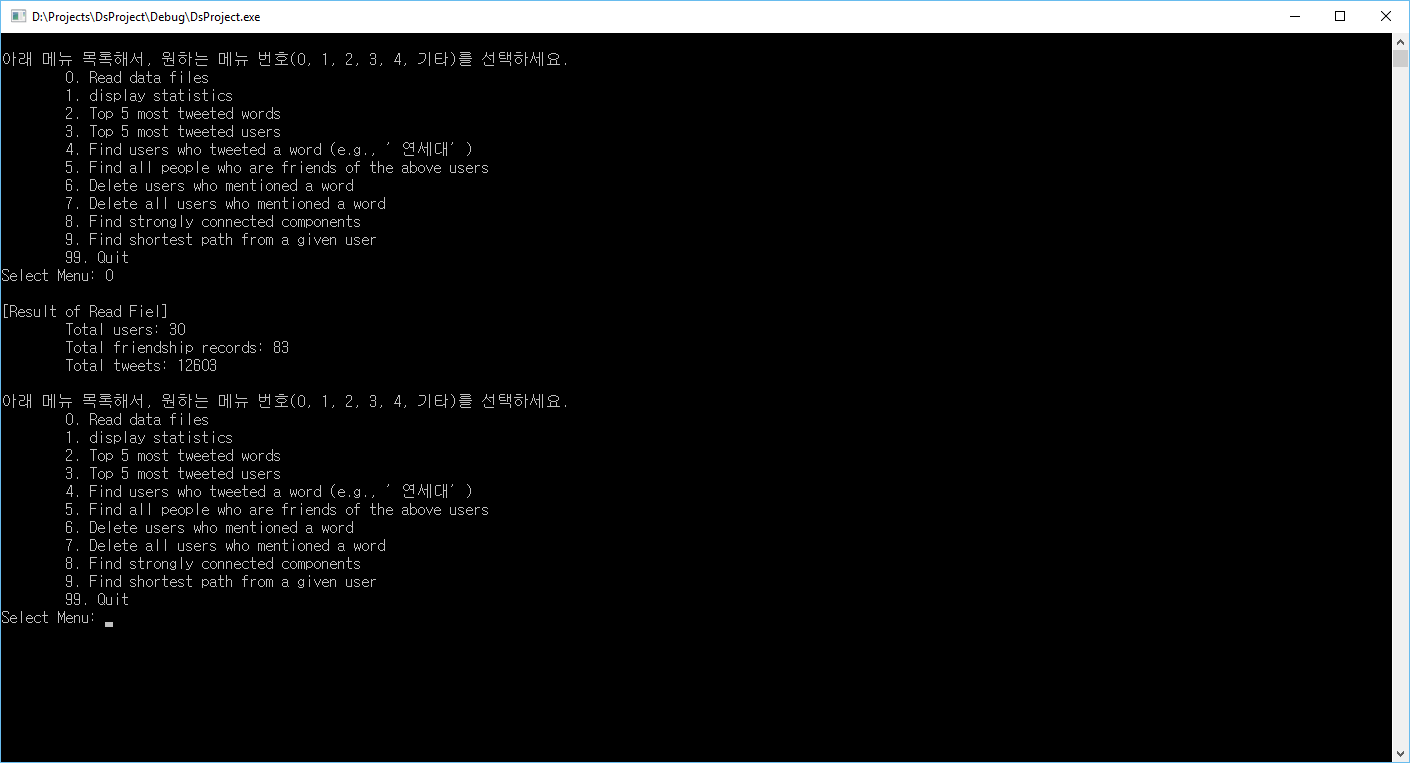
[**6.** **Find all users who are friend of the above user** 11](#_Toc453679070)

[**7.** **Top 5 strongly connected components** 12](#_Toc453679071)

[**8.** **Find shortest path from a user (id)** 14](#_Toc453679072)

# **Read data files**

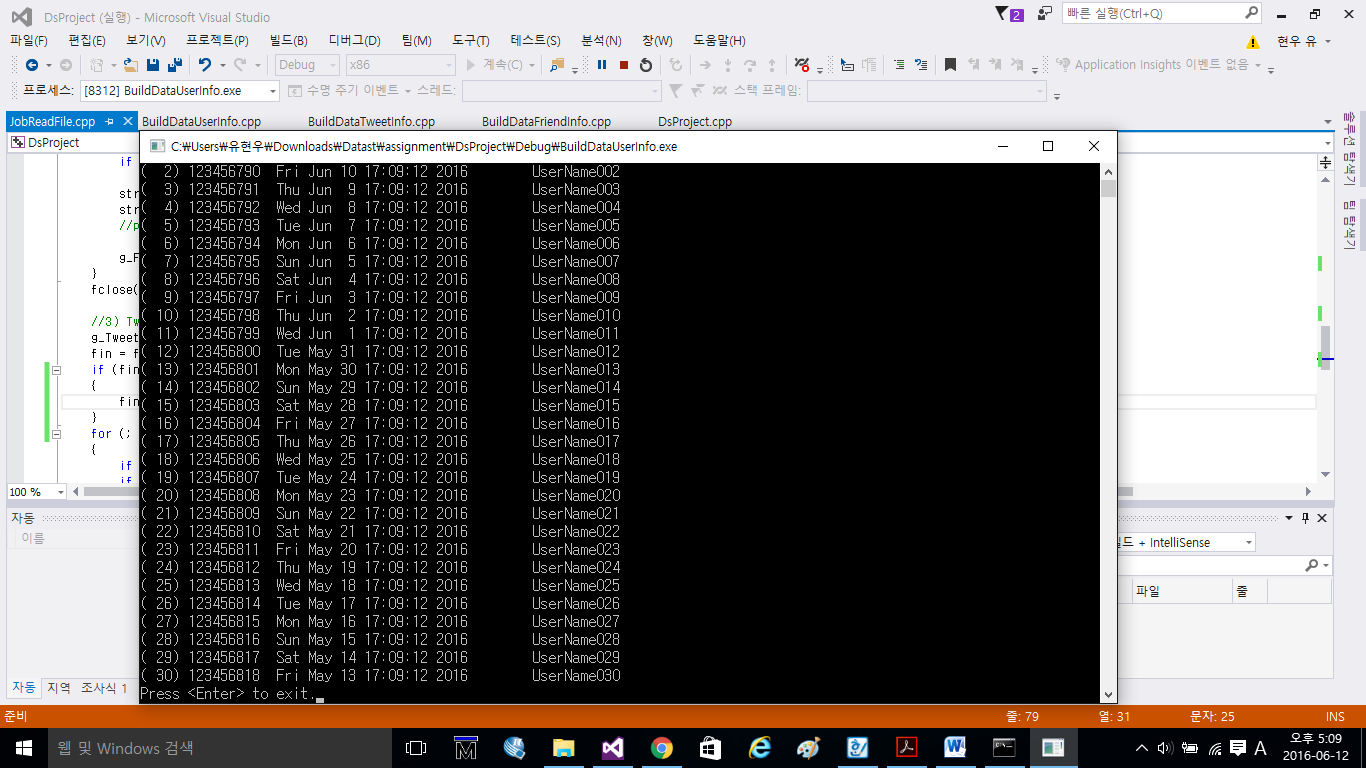
User, Friend, Tweet 데이터 파일의 내용일 읽어와서, 프로그램에서 사용하는 자료구조를 구성한다. 자료구조는 C 구조체(struct)에 대한 배열을 사용했다.



다양한 데이터를 효율적으로 적용할 수 있도록 샘플 데이터를 생성하는 프로그램을 작성해서 이용하고 있다.

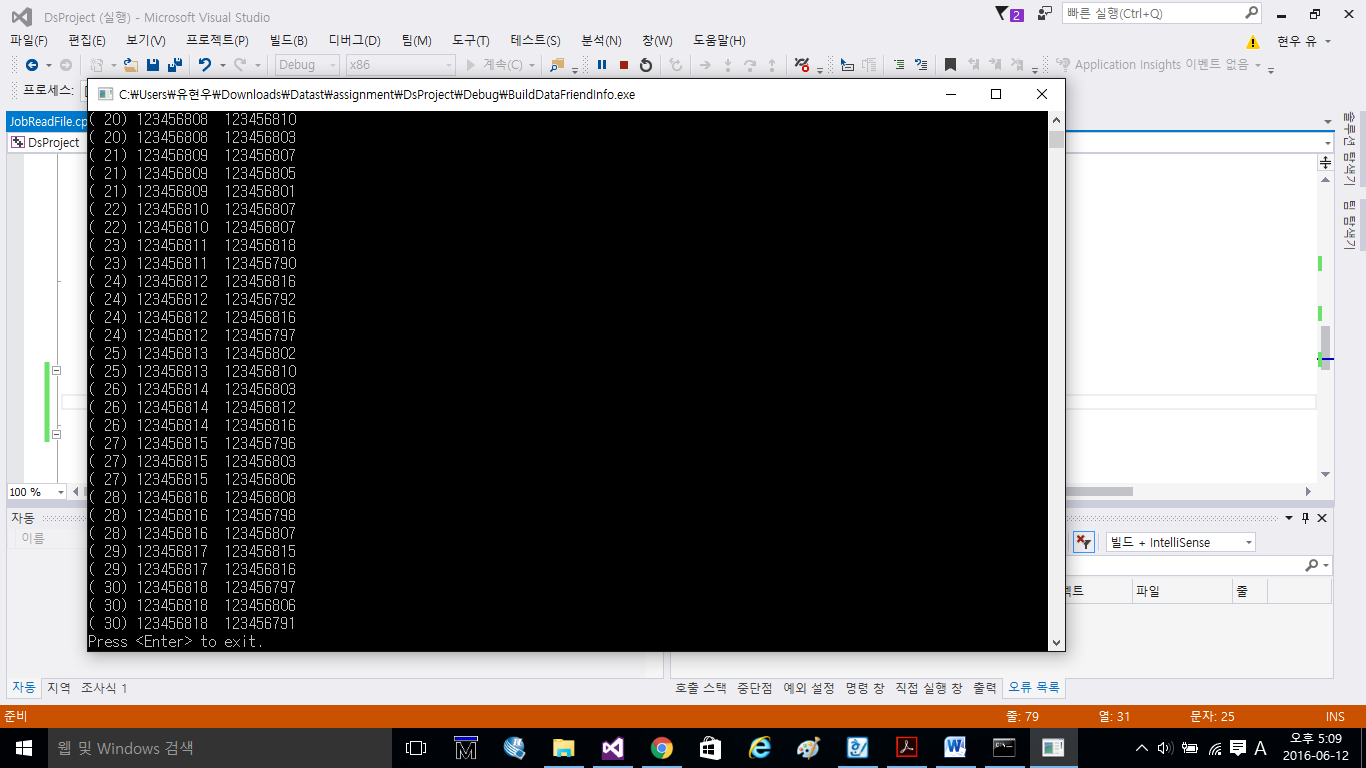
# **샘플 User 데이터 생성**

Visual Studio에서 Project BuildDataUserInfo를 시작 프로젝트로 설정하고 프로그램을 실행하거나, Command 창에서 BuildDataUserInfo.exe를 실행해서 샘플 User 데이터를 생성한다.



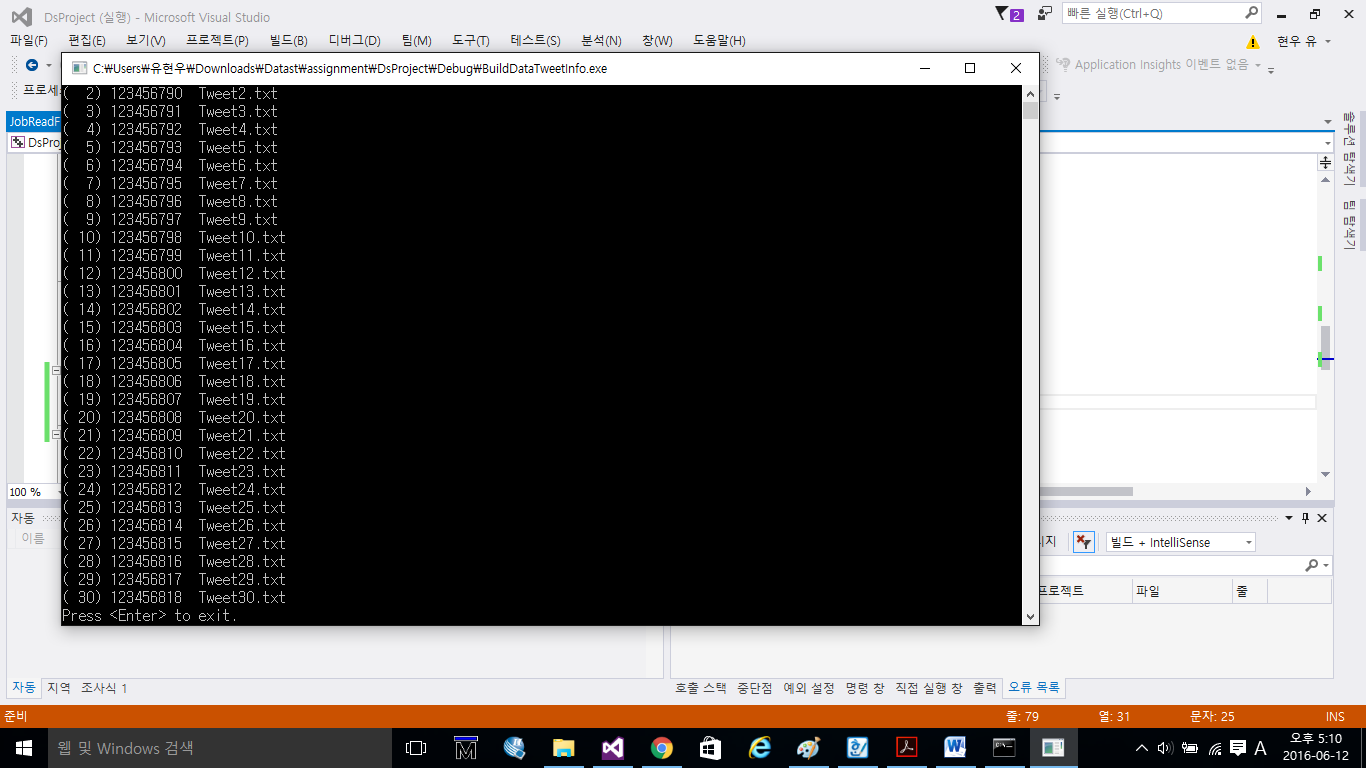
# **샘플 Friend 데이터 생성**

Visual Studio에서 Project BuildDataFriendInfo를 시작 프로젝트로 설정하고 프로그램을 실행하거나, Command 창에서 BuildDataFriendInfo.exe를 실행해서 샘플 Friend 데이터를 생성한다.



# **샘플 Tweet 데이터 생성**

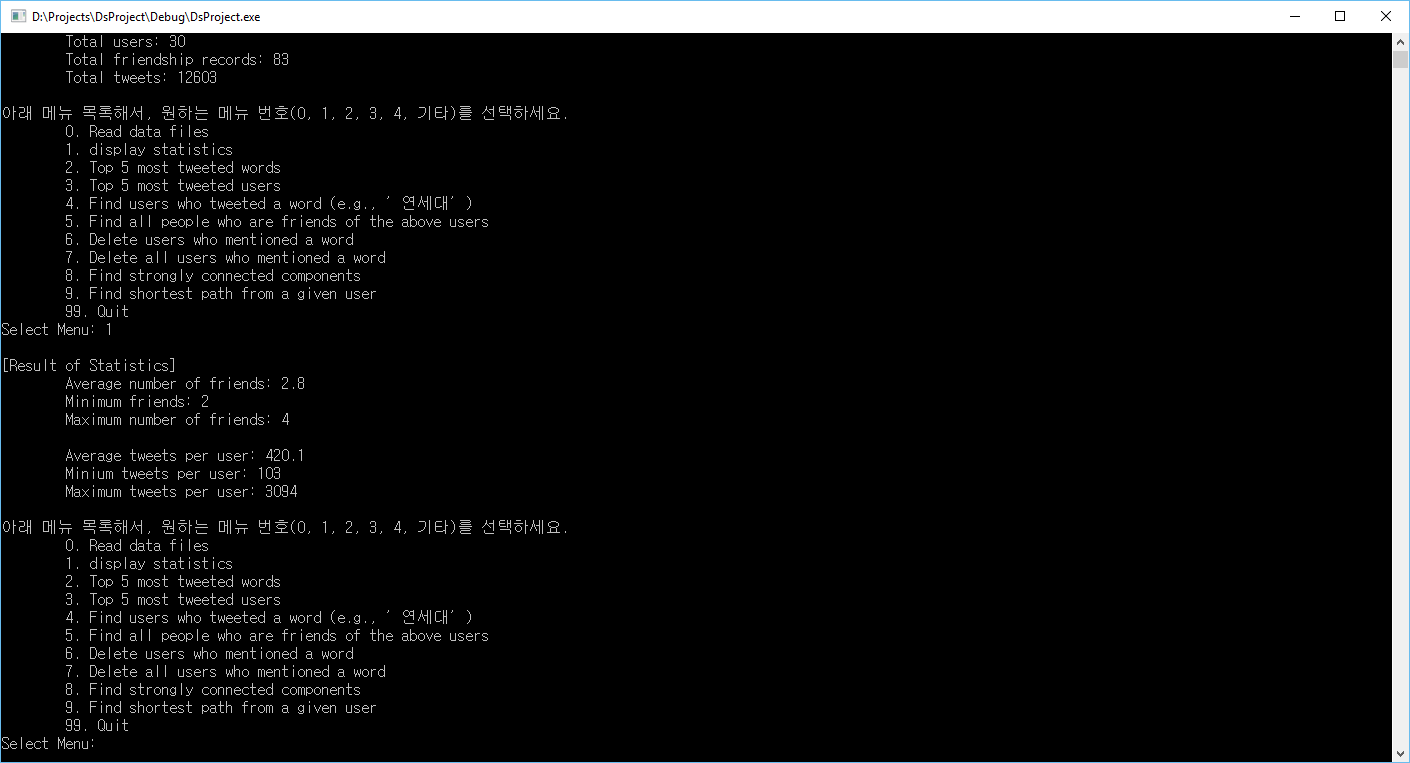
Visual Studio에서 Project BuildDataTweetInfo를 시작 프로젝트로 설정하고 프로그램을 실행하거나, Command 창에서 BuildDataTweetInfo.exe를 실행해서 샘플 Tweet 데이터를 생성한다.



# **Statistics**

파일에서 읽어 들인 Friend, Tweet 파일 내용을 분석해서, 통계자료를 추출한다. 이때 추출하는 통계자료는 다음과 같다. 작업 처리는 Linear Search를 사용해 O(n)의 Time Complexity로 작업을 처리했다.

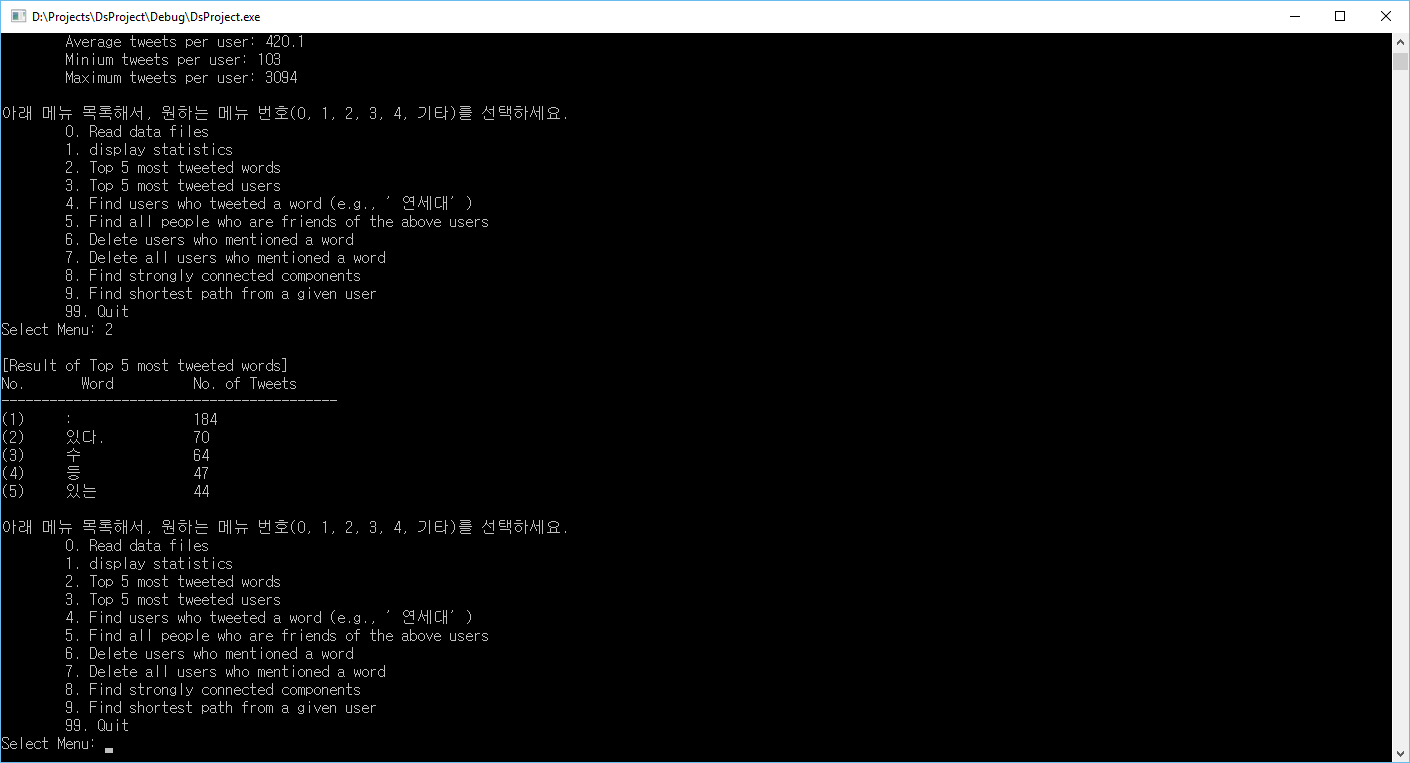
* Friend: Average # of friends, Minimum # of friends, Maximum # of friends.
* Friend: Average # of tweet words, Minimum # of tweet words, Maximum # of tweet words.



# **Top 5 most tweeted words**

Tweet 데이터에 대해 Linear Search를 적용해서, 각 Word별 발생 빈도에 대한 정보를 구성하고, Quick Sort를 이용해 발생 빈도를 기준으로 Tweet Word들을 정렬해서, 가장 발생 빈도가 높은 최상위 5 Word를 판별하고 있다. 따라서 각각 O(n)과 O(n log n)의 Time Complexity로 문제를 해결했다.

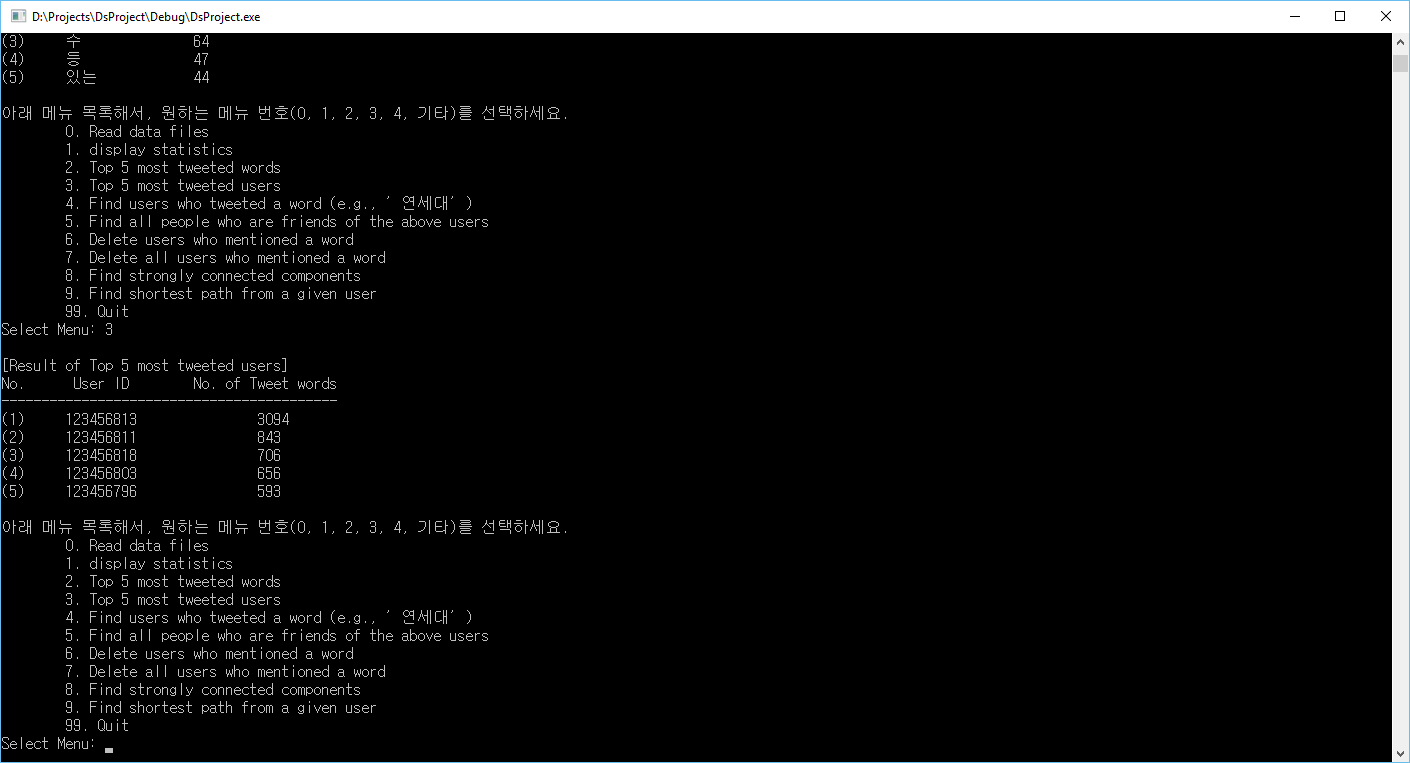
결과 화면에서는 최상위 순으로 가장 많은 발생 빈도를 나타내는 Word와 발생 빈도수를 출력하고 있다.



# **Top 5 most tweeted users**

Tweet 데이터에 대해 Linear Search를 적용해서, 각 User별 Tweet Word 개수에 대한 정보를 구성하고, Quick Sort를 이용해 Tweet Word 개수를 기준으로 User별 Tweet 정보들을 정렬해서, 가장 많은 Word를 Tweet한 최상위 5 User를 판별하고 있다. 따라서 각각 O(n)과 O(n log n)의 Time Complexity로 문제를 해결했다.

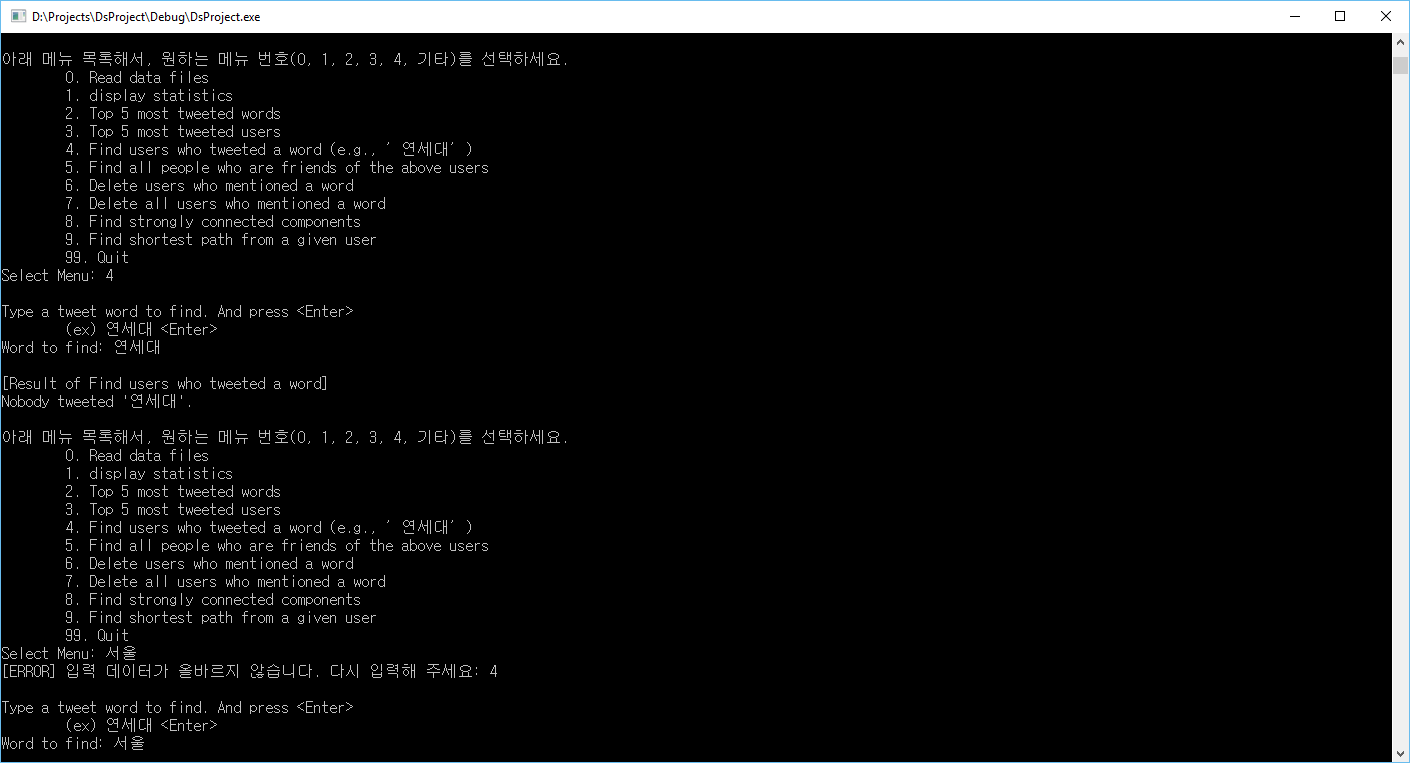
결과 화면에서는 가장 많은 Word를 Tweet한 User 순서로, User의 User ID와 Tweet한 Word 개수를 출력하고 있다.



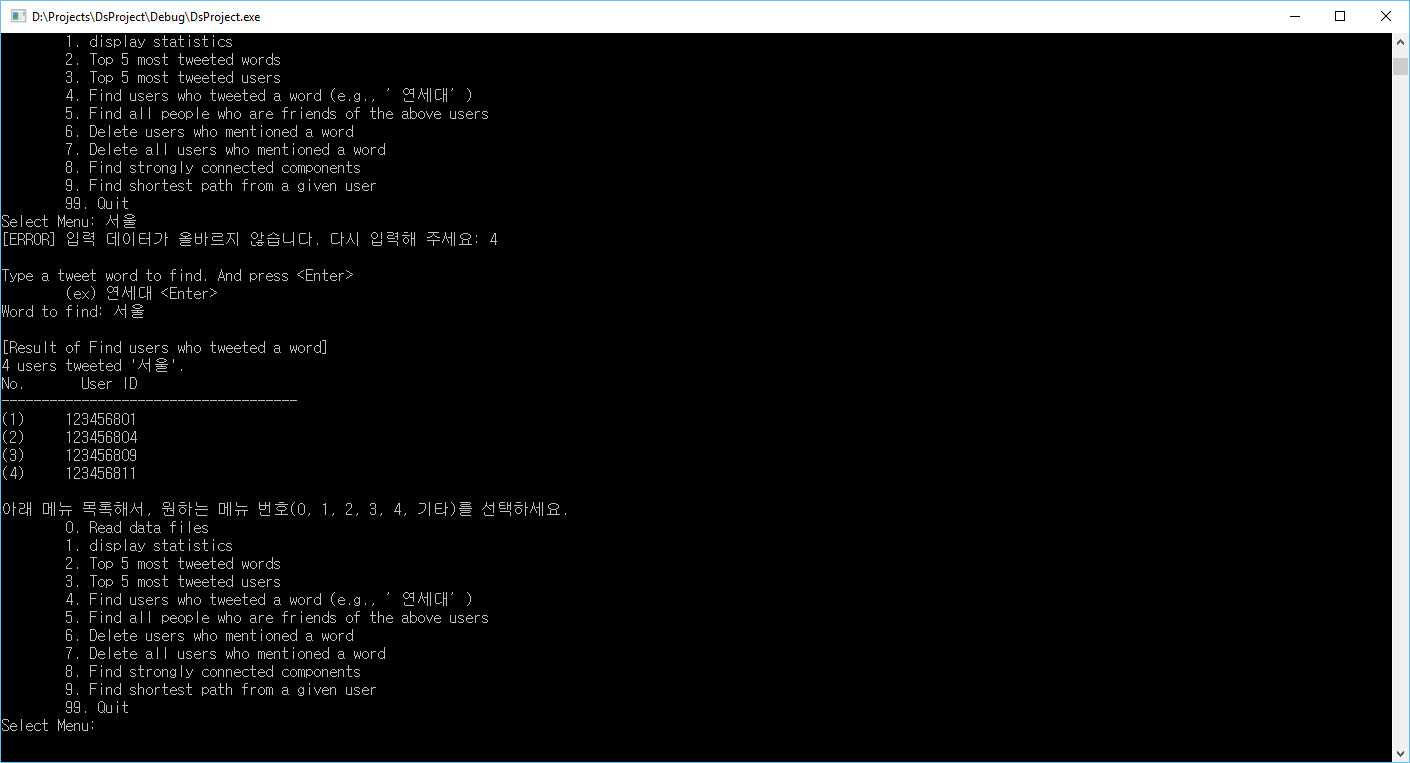
# **Find all users who mentioned a word**

사용자로부터 1개의 Keyword를 입력 받아, Linear Search를 이용해 Tweet 데이터 목록을 검사해서 사용자가 입력한 Keyword를 Tweet한 User들을 검색했다. 따라서 O(n) 의 Time Complexity로 문제를 해결했다.

만일 사용자가 입력한 Keyword를 Tweet한 User를 발견하지 못한 경우에는 메시지를 출력해서 이 사실을 알려주고 있다. 아래 결과 화면은 “연세대”라는 Word를 Tweet한 User들을 발견할 수 없다고 알려주고 있다.



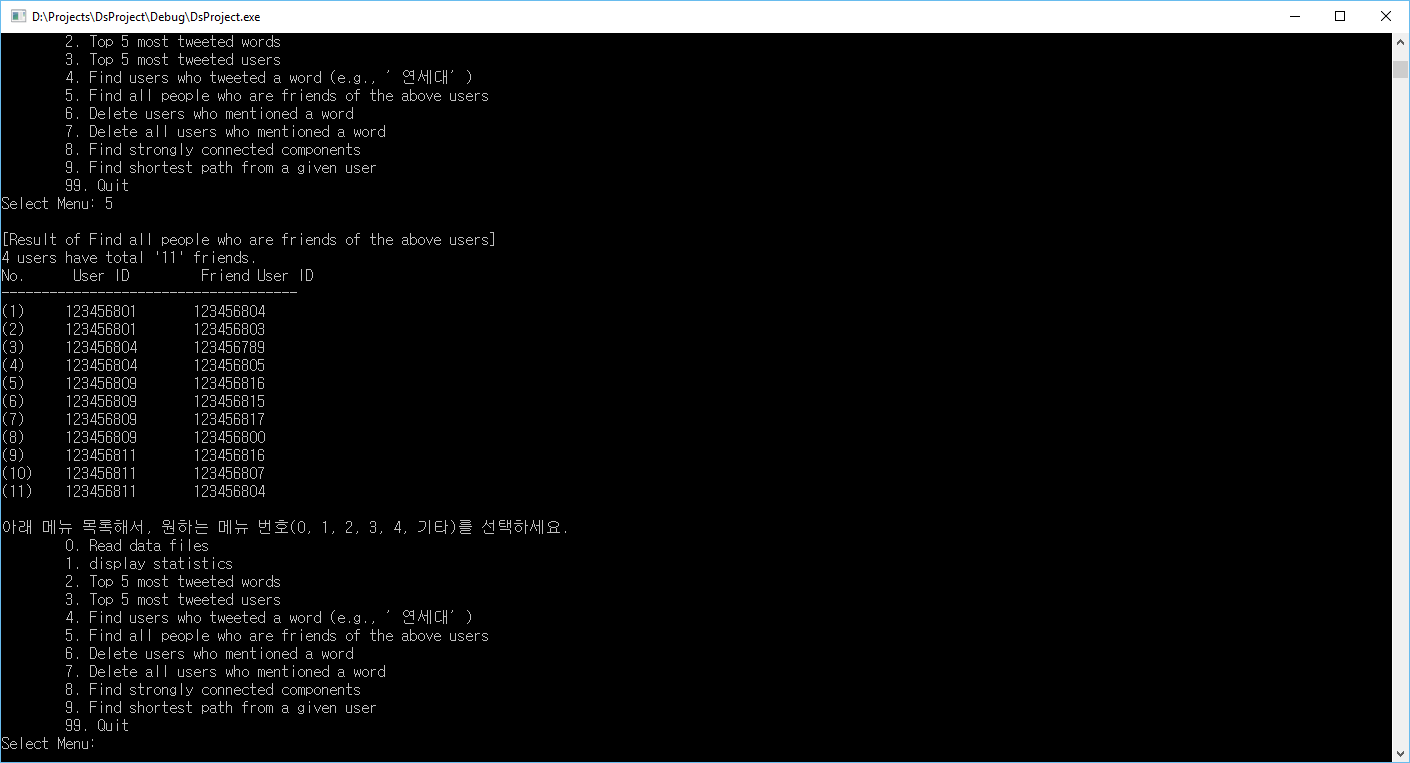
그리고 사용자가 입력한 Keyword를 Tweet한 User를 발견한 경우에는, 화면에 해당 User들의 User ID 목록을 출력하고 있다. 아래 결과 화면은 “서울”이라는 Word를 Tweet한 4명의 User 목록을 보여주고 있다.



# **Find all users who are friend of the above user**

Friend 데이터 목록에 대해 Linear Search를 이용해, 앞 단계에서 검색한 User 목록에 존재하는 User들이 관계를 가지고 있는 Friend 목록을 검색했다. 배열을 자료구조로 사용했기 때문에, O(n\*m) 의 Time Complexity로 문제를 해결했다.

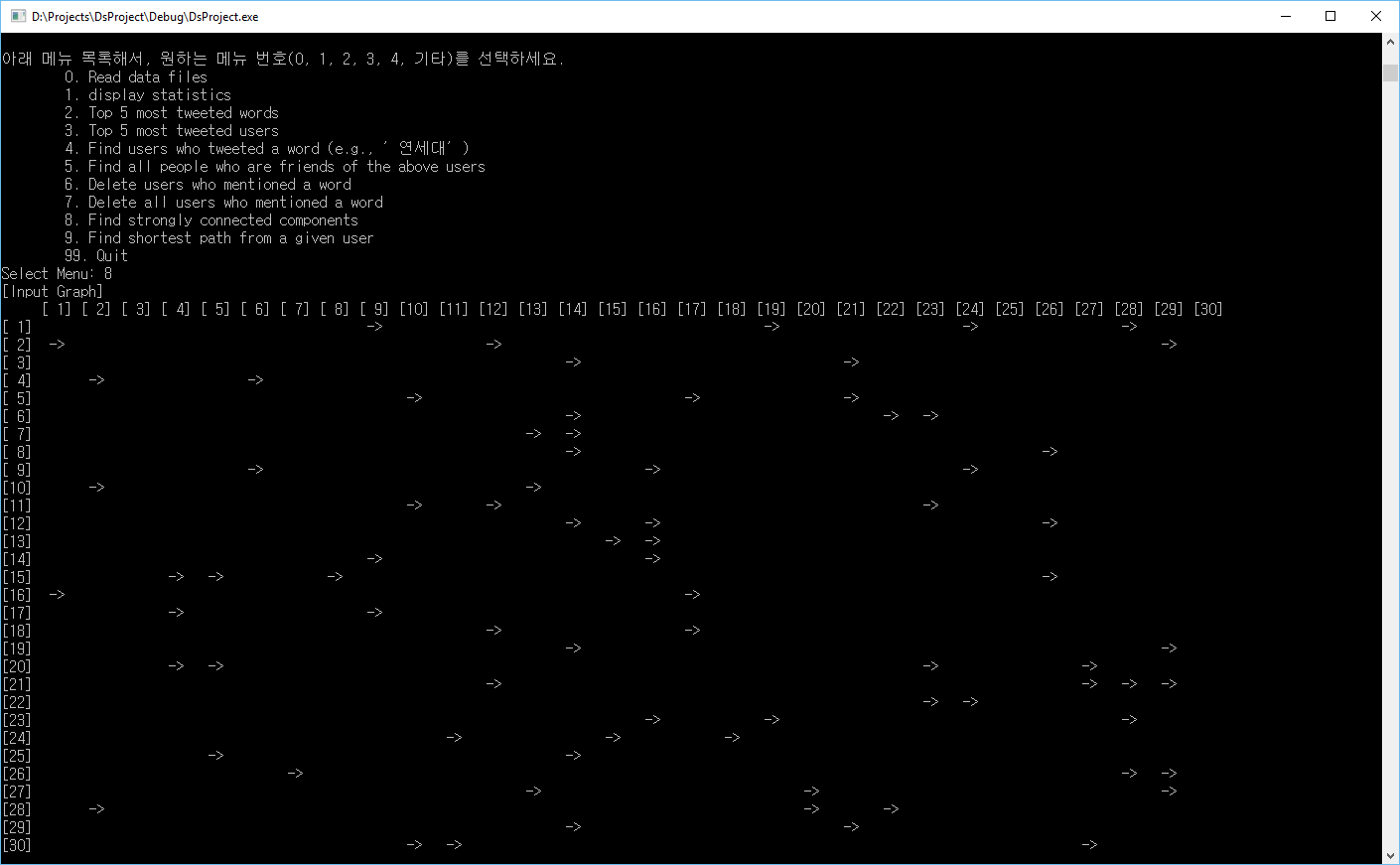
결과화면은 앞 단계에서 검색한 각각의 User에 대해, 해당 User가 Friend 관계를 가지고 있는 User에 대한 User ID 목록을 출력하고 있다.



# **Top 5 strongly connected components**

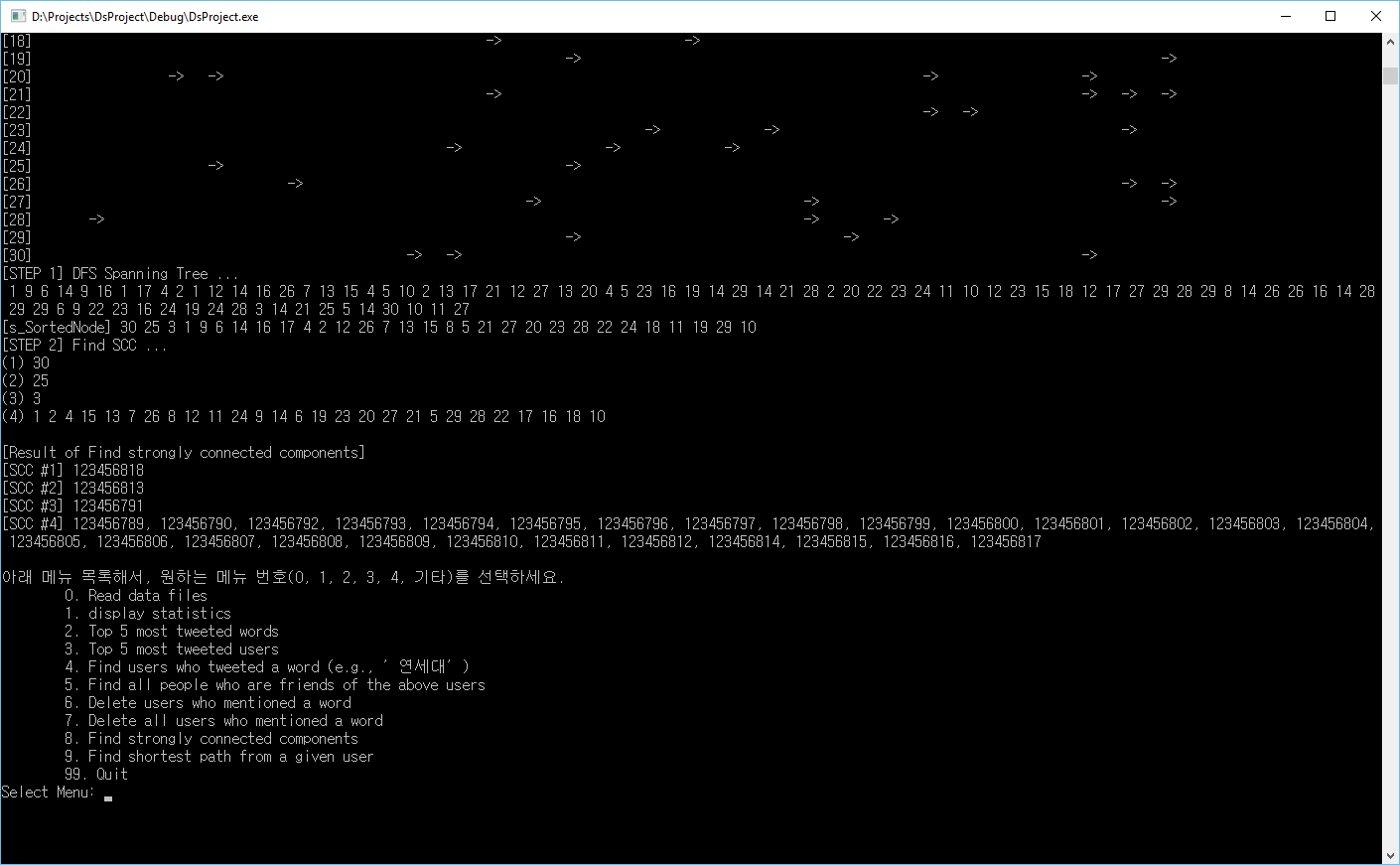
자료구조 Graph에서 Node는 User에 해당하고, User들 사이에 Friend 관계가 있는 경우에 Directed Edge가 존재한다. Friend 관계를 정의하는 Graph를 구성하기 위해 2차원 배열 Matrix를 자료구조로 사용했다. Graph[i][j]는 i라는 User가 j라는 User를 Friend로 하고 있는 경우에 1이라는 값을 가지고, Friend가 아닌 경우에는 0이라는 정보를 가진다.

일단 현재 사용하는 데이터에 대한 Friend 관계를 화면에 출력하고 있다. 화면에서 y축인 j를 의미하고, x축은 i를 의미한다. 따라서 아래 화면의 경우 User 1은 User 9, 19, 24, 28을 Friend로 하고 있다는 것을 알 수 있다.



2차례의 DFS(Depth First Search)를 사용해서 Strongly Connected Component를 추출하고 있다. 첫 번째 DFS에서는 Spanning Tree를 구성해서 각 Node에 Sort No.(이후에 Strongly Connected Component를 찾기 위해 방문하는 순서의 역순)를 할당한다.

그리고 Sort No. 역순으로 DFS를 사용해, Strongly Connected Component들을 판별한다. 아래 결과화면은 모두 4개의 Strongly Connected Component가 존재하고 있음을 알려주고 있다. 3명의 User들은 자신만이 고립되어 있고, 이들을 제외한 모든 User들은 동일한 Strongly Connected Component에 포함되어 있음을 보여주고 있다.

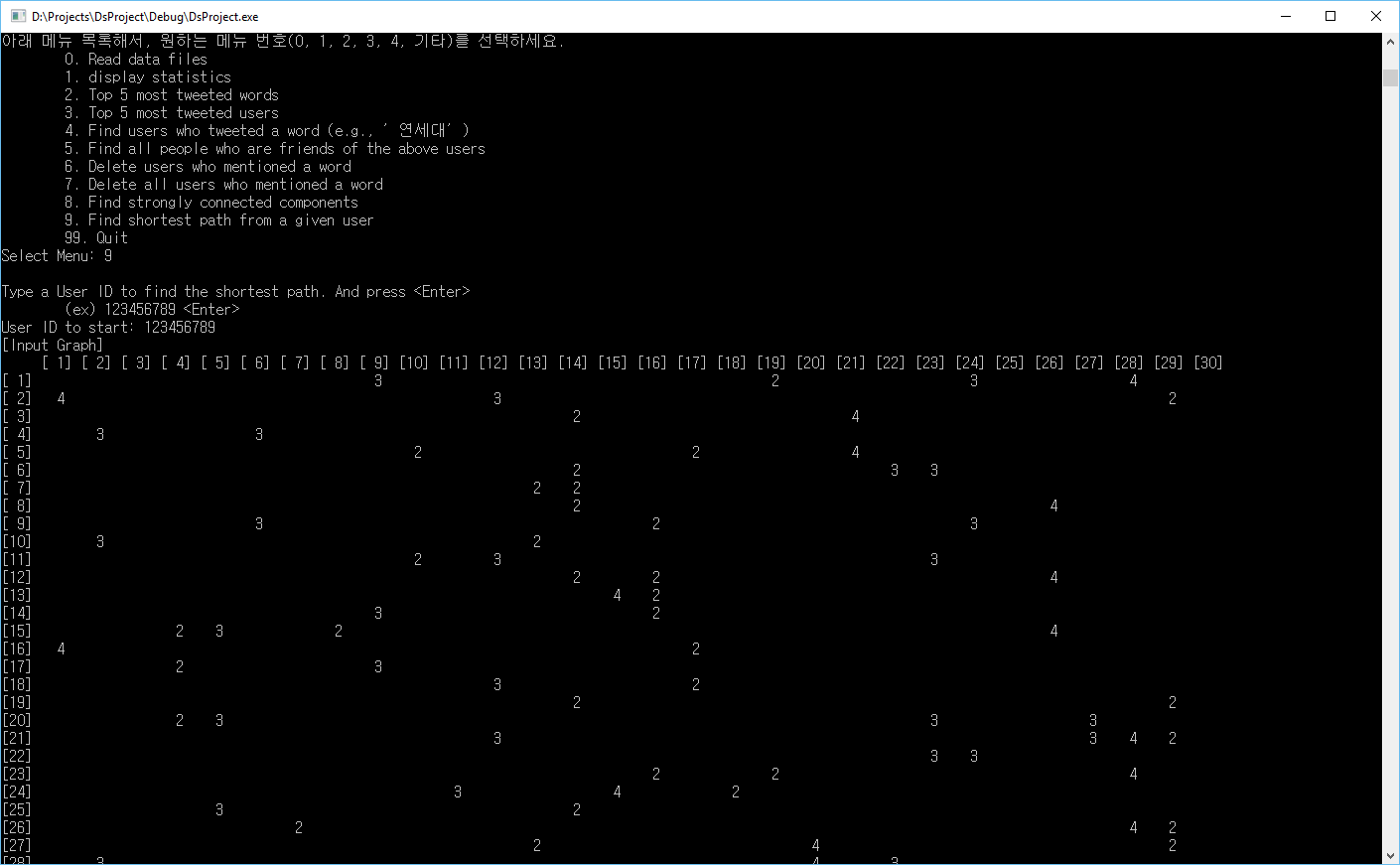


# **Find shortest path from a user (id)**

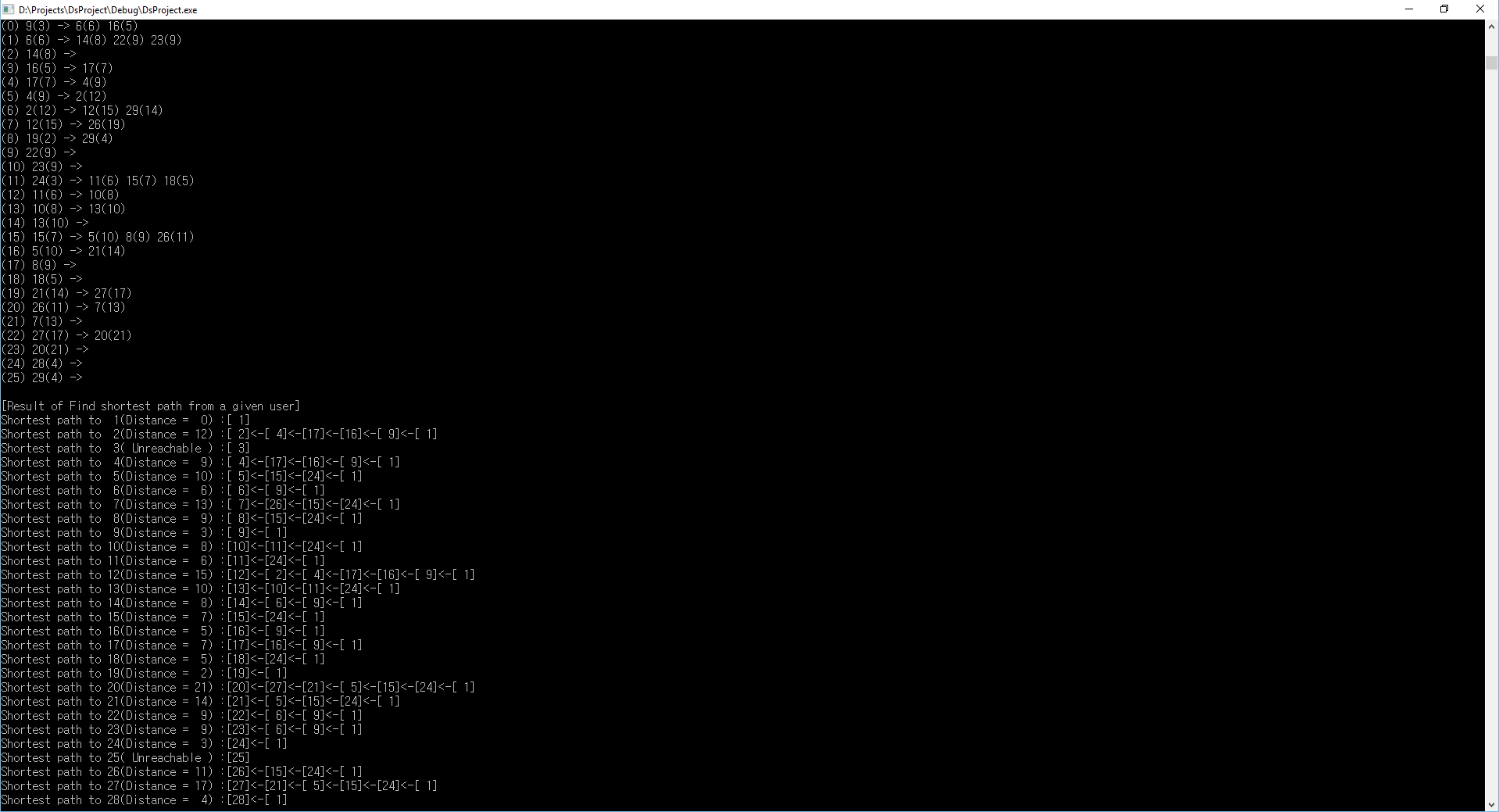
사용자로부터 시작 위치에 해당하는 User ID를 입력 받아, 주어진 User로부터 시작해서 다른 모든 User들에 도달하는 Shortest Path를 찾아낸다.

이때 자료구조 Graph에서 Node는 User에 해당하고, User들 사이에 Friend 관계가 있는 경우에 Directed Edge가 존재한다. 그리고 Directed Edge의 weight는 Friend 수를 사용했다. Friend 관계를 정의하는 Graph를 구성하기 위해 2차원 배열 Matrix를 자료구조로 사용했다. Graph[i][j]는 i라는 User가 j라는 User를 Friend로 하고 있는 경우에 j라는 User에 존재하는 Friend 수를 값을 가지고, Friend가 아닌 경우에는 0이라는 정보를 가진다.

먼저 사용자로부터 Shortest Path를 추출하는 시작 위치에 해당하는 User ID를 입력 받는다. 다음에는 Friend 정보를 조사해서 이차원 배열을 자료구조로 사용하는 Graph를 구성해서 화면에 출력한다. 화면에서 y축인 j를 의미하고, x축은 i를 의미한다. 따라서 아래 화면의 경우, User 1은 User 9, 19, 24, 28을 Friend로 하고 있다는 것을 알 수 있다. 그리고 User 9, 19, 24, 28는 각각 3, 2, 3, 4명의 Friend를 가지고 있다는 것을 의미한다.



사용자가 입력한 User에 해당하는 Node로부터 시작해서, Shortest Path를 추출한다. Shortest Path 추출을 위해, 검사를 수행하는 Node들의 작업순서가 화면에 출력되고 있다. 즉, 아래 화면에서는 시작 위치에 해당하는 User 1번부터 시작해서 9 -> 6 -> 14 -> 16 -> … 순서로 작업을 진행하고 있음을 알 수 있다.



이처럼 작업을 진행해서 완료한 다음에는, 각 User까지 도달하기 위한 Shortest Path의 경로와 Distance를 출력하고 있다. 예를 들어 위의 결과 화면에는 User 1에서 User 2에 도달하기 위해 “1 -> 9 -> 16 -> 17 -> 4 -> 2”라는 Path를 Shortest Path로 가지고, Distance는 12라는 것을 보여주고 있다. 그리고 User 3의 경우에는 User 1로부터 도달하는 Path가 존재하지 않는다고 보여주고 있다.