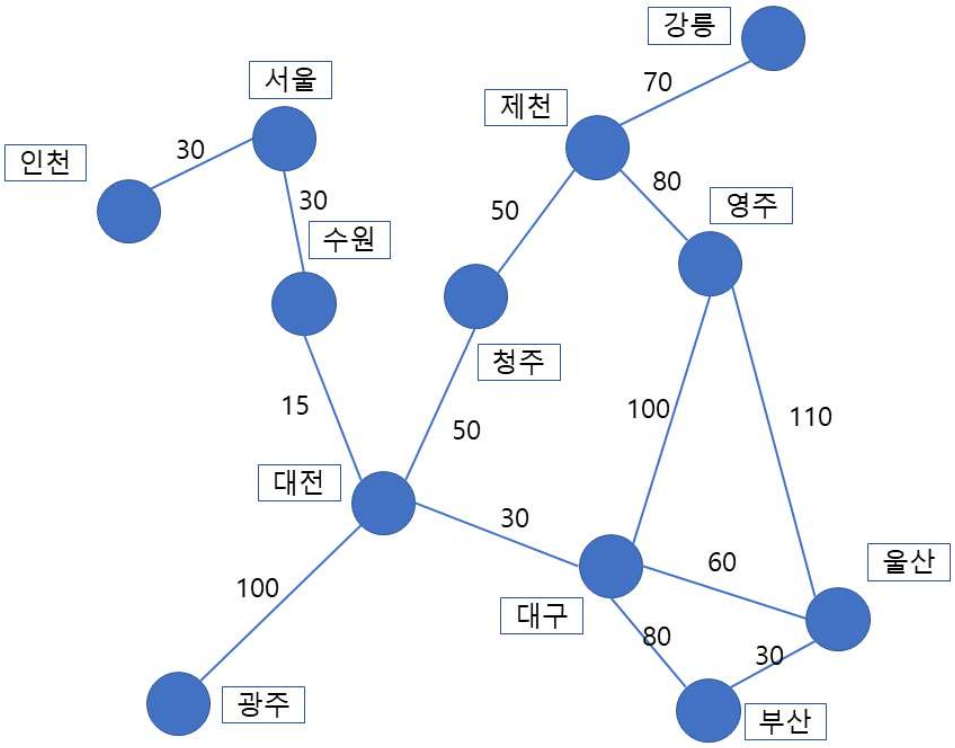
Data Structure Lab. Project #3

2018년 11월 14일

Due date: 2018년 12월 12일 수요일 23:59:59

본 프로젝트에서는 도로 그래프를 이용해 최단경로 찾기 프로그램을 구현한다. 이 프로그램은 도시를 연결하는 도로에 대한 정보가 저장되어있는 텍스트 파일을 통해 그래프를 구현하고, DFS, Dijkstra, Bellman-Ford 알고리즘을 이용해 도로의 경로와 거리를 구한다. 도시의 도로 그래프 정보는 방향성(Direction)과 가중치(Weight)를 모두 가지고 있으며, Matrix 형태로 저장되어있다. 만약 Weight가 음수일 경우 DFS와 Dijkstra는 에러를 출력하며, Bellman-Ford에서는 음수 사이클이 발생한 경우 에러, 음수 사이클이 발생하지 않았을 경우 최단 경로와 거리를 구한다. FLOYD에서는 음수 사이클이 발생한 경우 에러, 음수 사이클이 발생하지 않았을 경우 최단 경로 행렬을 구한다. 프로그램의 동작은 명령어 파일에서 요청하는 명령에 따라 각각의 기능을 수행하고 그 결과를 로그(log.txt) 파일에 저장한다.



[그림 1] 도시 최단경로 찾기 예시

□ Program implementation

1. 도로 정보 데이터

프로그램은 방향성과 가중치를 가지고 있는 그래프 정보를 Matrix 형태로 도로 정보 데이터를 저장한 텍스트 파일을 LOAD 명령어를 통해 읽어 해당 정보를 Graph 클래스에 저장한다. 도로 정보 데이터 텍스트 파일의 첫 번째 줄에는 도시의 개수가 저장되어있고 다음 줄에 도로 정보 데이터가 저장되어있다. 도로 정보 데이터의 행과 열은 각각 Edge의 Start Vertex와 End Vertex의 Weight를 의미한다(표 1). 이때, 도로 정보 데이터의 모든 Vertex 값은 0 이상의 정수이며, 0부터 시작하여 0, 1, 2, 3, 4 ... 순으로 정해진다. Start Vertex와 End Vertex가 연결되지 않은 경우 Weight가 0, 연결된 경우 Weight가 0이 아닌 값을 가진다(그림 2). 연결된 도시의 길은 같은 길이어도 장애물로 인해 진행 방향에 따라 Weight가 다를 수 있으며, Weight가 음수인 길이 존재할 수도 있다.

줄 수	내용
1	그래프 크기(n)
2 ... n+1	[Weight_1] [Weight_2] [Weight_3] ... [Weight_n]

표 1. 도로 정보 데이터 형식

```
5
0 6 13 0 0
0 0 5 6 0
2 0 0 7 4
0 6 0 0 3
0 0 5 2 0
```

그림 2. 도로 정보 데이터가
저장되어있는 텍스트 파일의 예

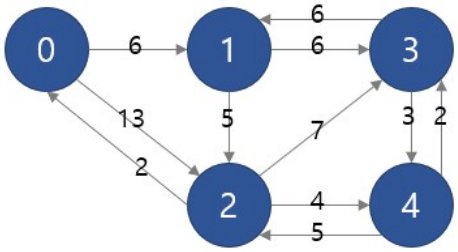


그림 3. 그림 2의 맵 연결 예

Vertex와 Edge는 Linked-List를 이용하여 연결하며, 맵 데이터를 순차적으로 읽어 Linked-List의 가장 끝에 연결한다(순차적으로 연결하여 반드시 Vertex 오름차순으로 정렬된 형태로 저장되어 있어야 한다).

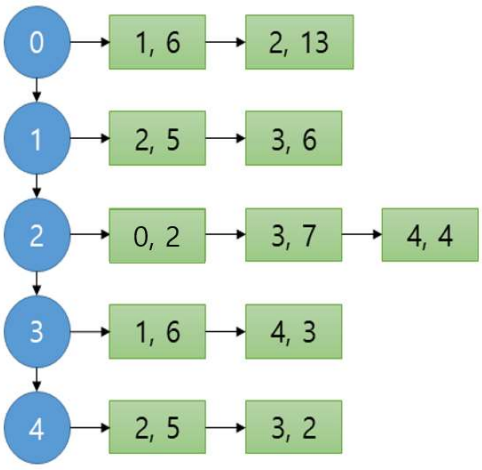


그림 4. Linked-List 연결 예시

2. 그래프 연산

프로그램은 텍스트 파일로부터 도로 정보 데이터를 읽은 뒤, 명령어에 따라 알맞은 그래프 연산을 수행할 수 있다.

- DFS

  DFS 명령어는 Start Vertex와 End Vertex를 입력받아 경로와 거리를 구한다. 경로를 구하는 방법은 재귀적 호출 방법과 Stack을 사용한 방법이 있으며, 본 프로그램에서는 Stack을 사용(직접 구현)하여 구하도록 한다. Stack을 사용할 때 값이 가장 작은 인접 Vertex부터 순차적으로 삽입한다. Stack에는 동일한 Vertex가 삽입되는 경우가 있을 수 있으며, 한번 탐색한 경로를 다시 탐색하지는 않는다.

- DIJKSTRA

  DIJKSTRA 명령어는 Start Vertex와 End Vertex를 입력받아 최단 경로와 거리를 구한다. Dijkstra 연산을 위해 STL set을 이용하여 최단 거리와 경로를 구하고, 결과로 찾은 최단 경로를 다시 순회하여 Start Vertex부터 End Vertex 순으로 경로를 구한다.

- DIJKSTRAMIN

  DIJKSTRAMIN 명령어는 DIJKSTRA 명령어와 동작이 동일하며, Dijkstra 연산을 위해 STL set이 아닌 Min-Heap을 직접 구현하여 최단 거리와 경로를 구한다. (MinHeap.h 참조)

- BELLMANFORD

  BELLMANFORD 명령어는 Start Vertex와 End Vertex를 입력받아 최단 경로와 거리를 구한다. Weight가 음수인 경우에도 정상 동작하며, 음수 Weight에 사이클이 발생한 경우에는 알맞은 에러를 출력한다.

- FLOYD

  FLOYD 명령어는 입력 인자 없이 모든 쌍에 대해서 최단 경로 행렬을 구한다. Weight가 음수인 경우에도 정상 동작하며, 음수 Weight에 사이클이 발생한 경우에는 알맞은 에러를 출력한다.

3. 정렬 연산

정렬 연산은 최단 경로로 방문한 노드를 정렬하여 출력한다. 연산을 향상시키기 위해 segment size에 따라 효율적인 정렬 알고리즘을 구현한다(강의자료 13장 Sorting 참고). 해당 프로젝트에서는 퀵정렬 수행시 segment size를 재귀적으로 분할할 때, segment size가 6이하일 경우 삽입정렬을 수행한다. segment size가 7이상인 경우는 계속해서 분할한다. 정렬은 제공된 의사코드에 기반하여 작성한다.

STL sort Algorithm

```
1  void quickSort(arr, low, high )
2     if(low < high)
3        if(high–low+1 <= segment_sz)
4           insertionSort(arr, low, high)
5        else
6           pivot = partition(a, low, high)
7           quickSort(a, low, pivot-1)
8           quickSort(a, pivot+1, high)
```

그림 5. STL sort 의사코드

□ Program implementation

프로그램은 명령어를 통해 동작하며, 실행할 명령어가 작성되어있는 커맨드 파일(command.txt)을 읽고 명령에 따라 순차적으로 동작한다. 각 명령어의 사용법과 기능은 다음과 같다.

명령어	명령어 사용 예 및 기능
LOAD	사용법) LOAD textfile ✓ 프로그램에 도로 정보 데이터를 불러오는 명령어로, 도로 정보 데이터가 저장되어있는 파일을 읽어 그래프를 구성한다. ✓ 텍스트 파일이 존재하지 않을 경우, 로그 파일(log.txt)에 오류 코드(101)를 출력한다. ✓ 하나의 커맨드 파일에서 LOAD가 2번 이상 성공하는 경우는 고려하지 않는다.
PRINT	사용법) PRINT ✓ 도로 정보 데이터를 읽어 도로 정보를 출력하는 명령어로, Matrix 형태로 도로 정보를 출력한다. ✓ 도로 정보 데이터가 존재하지 않을 경우, 로그 파일(log.txt)에 오류 코드(202)를 출력한다.

DFS	사용법) DFS StartVertex EndVertex 예시) DFS 0 3 ✓ StartVertex를 기준으로 DFS를 수행하여 EndVertex까지의 경로와 거리를 구하는 명령어로, DFS 결과를 로그 파일(log.txt)에 저장한다. ✓ DFS를 수행하고 경로를 StartVertex부터 EndVertex까지 순서대로 거리와 함께 저장한다. ✓ 입력한 Vertex가 그래프에 존재하지 않거나, 입력한 Vertex가 부족한 경우, 또는 음수인 Weight가 있는 경우 로그 파일(log.txt)에 알맞은 오류 코드를 저장한다.
DIJKSTRA	사용법) DIJKSTRA StartVertex EndVertex 예시) DIJKSTRA 0 3 ✓ StartVertex를 기준으로 Dijkstra를 수행하여 EndVertex까지의 최단 경로와 거리를 구하는 명령어로 STL set을 이용하여 구현하며, Dijkstra 결과를 로그 파일(log.txt)에 저장한다. ✓ Dijkstra를 수행하고 최단 경로를 StartVertex부터 EndVertex까지 순서대로 최단 거리와 함께 저장한다. ✓ 입력한 Vertex가 그래프에 존재하지 않거나, 입력한 Vertex가 부족한 경우, 또는 음수인 Weight가 있는 경우 로그 파일(log.txt)에 알맞은 오류 코드를 저장한다.
DIJKSTRAMIN	사용법) DIJKSTRAMIN StartVertex EndVertex 예시) DIJKSTRAMIN 0 3 ✓ StartVertex를 기준으로 Dijkstra를 수행하여 EndVertex까지의 최단 경로와 거리를 구하는 명령어로 Min-Heap을 직접 구현하며, Dijkstra 결과를 로그 파일(log.txt)에 저장한다. ✓ Dijkstra를 수행하고 최단 경로를 StartVertex부터 EndVertex까지 순서대로 최단 거리와 함께 저장한다. ✓ 입력한 Vertex가 그래프에 존재하지 않거나, 입력한 Vertex가 부족한 경우, 또는 음수인 Weight가 있는 경우 로그 파일(log.txt)에 알맞은 오류 코드를 저장한다.
BELLMANFORD	사용법) BELLMANFORD StartVertex EndVertex 예시) BELLMANFORD 0 3 ✓ StartVertex를 기준으로 Bellman-Ford를 수행하여 EndVertex까지의

	최단 경로와 거리를 구하는 명령어로, Bellman-Ford 결과를 로그 파일(log.txt)에 저장한다. ✓ 음수인 Weight가 있는 경우에도 동작해야 한다. ✓ 입력한 Vertex가 그래프에 존재하지 않거나, 입력한 Vertex가 부족한 경우, 또는 음수인 Weight를 가지는 사이클이 발생한 경우 로그 파일(log.txt)에 알맞은 오류 코드를 저장한다.
FLOYD	사용법) FLOYD ✓ 모든 도시의 쌍 (StartVertex, EndVertex)에 대해서 도시 StartVertex에서 EndVertex로 가는데 필요한 비용의 최솟값을 행렬 형태로 저장한다. ✓ FLOYD의 결과를 로그 파일(log.txt)에 저장한다.

표 2. 명령어 기능 및 사용 예

□ Requirements in implementation

- ✓ 모든 명령어는 커맨드 파일에 저장하여 순차적으로 읽고 처리한다.
  - 커맨드 파일 이름을 “command.txt”로 절대 고정하지 않는다.(반드시 Run 함수의 인자인 filepath를 사용)
- ✓ 명령어와 함께 입력되는 Vertex는 숫자로 입력한다.
- ✓ 명령어에 인자(Parameter)가 부족한 경우 오류 코드를 출력한다.
- ✓ 예외처리에 대해 반드시 오류 코드를 출력한다.
- ✓ 출력은 “출력 포맷”을 반드시 따라한다.
- ✓ 로그 파일(log.txt)에 출력 결과를 반드시 저장한다.
  - 에러가 발생했을 경우 그에 맞는 에러 명을 출력한다.
  - 프로그램 실행 시 로그 파일(log.txt)이 이미 존재할 경우 이전 내용을 모두 지우고 시작한다.
- ✓ 로그 파일(log.txt)에 에러 결과를 반드시 저장한다.
  - 에러가 없는 경우 0(Success), 있는 경우 그에 맞는 에러 코드를 출력한다. (모든 명령어에 에러 코드가 출력되어야 한다.)
  - 프로그램 실행 시 로그 파일(log.txt)이 이미 존재할 경우 이전 내용을 모두 지우고 시작한다.
- ✓ 프로그램이 종료될 때 메모리 누수가 발생하지 않도록 한다.

□ Error Code

명령어에 인자가 부족한 경우, 또는 각 상황마다 오류 코드를 출력해야하는 경우, 로그 파일(log.txt)에 알맞은 에러 코드 및 에러 명을 출력한다. 즉, 모든 명령어 동작에 에러 코드가 로그 파일(log.txt)에 출력되어야 하며, 에러 명을 로그 파일(log.txt)에 출력해야 한다. 제공되는 Result.h를

반드시 사용하며, 표 3에 명시된 에러 코드 이외의 문제는 따로 고려하지 않는다. 이외의 내용은 Requirements in implementation을 참조한다. 각 명령어별 오류 코드는 다음과 같다.

명령어	에러 코드	내용
LOAD	101	에러 명: LoadFileNotExist - 도로 정보 데이터 텍스트 파일이 존재하지 않을 경우
PRINT DFS DIJKSTRA DIJKSTRAMIN BELLMANFORD FLOYD	202	에러 명: GraphNotExist - 도로 정보 데이터가 존재하지 않을 경우
DFS DIJKSTRA DIJKSTRAMIN BELLMANFORD	200	에러 명: VertexKeyNotExist - 명령어 인자(Vertex)가 부족한 경우
	201	에러 명: InvalidVertexKey - 인자로 입력한 Vertex가 도로 정보 데이터에 없는 경우
DFS DIJKSTRA DIJKSTRAMIN	203	에러 명: InvalidAlgorithm - 도로 정보 데이터에 음수인 Weight가 존재할 경우
BELLMANFORD	204	에러 명: NegativeCycleDetected - 도로 정보 데이터에 Weight가 음수인 사이클이 존재할 경우
ETC	0	에러 명: Success - 표 2의 명령어 수행 시 에러가 발생하지 않은 경우 - 결과 출력이 없는 명령어일 경우, 위의 에러 명 출력(LOAD만 해당)
	100	에러 명: CommandFileNotExist - run 함수의 인자로 입력받은 커맨드 파일이 존재하지 않을 경우 - 결과 파일(result.log)에서 에러 명령어 자리에 “SYSTEM”을 사용 - 커맨드 파일 이름을 “command.txt”로 절대 고정해서 사용하지 않음

		- ex) ====== SYSTEM ====== CommandFileNotExist ====================
	300	에러 명: NonDefinedCommand - 존재하지 않는 명령어일 경우 - 결과 파일(log.txt)에서 에러 명령어 자리에 오류가 발생한 명령어를 사용 - ex) ASTAR ====== ASTAR ====== NonDefinedCommand ====================

표 3. 에러 코드

□ Print Format

각 명령어 동작에 따라 로그 파일(log.txt)에 해당 내용을 지정된 형식에 맞추어 출력한다. 이외의 내용은 Requirements in implementation을 참조한다. 각 명령어별 출력 형식은 다음과 같다.

기능	출력 포맷
LOAD	====== LOAD ====== Success ==================== ==================== Error code: 0 ====================
PRINT	====== PRINT ====== 0 6 13 0 0 0 0 5 6 0 2 0 0 7 4 0 6 0 0 3 0 0 5 2 0 ==================== ==================== Error code: 0

	========== ========== ========== ========== ========== ========== ========== ========== ========== ==========
DFS	===== DFS ===== shortest path: 0 2 4 3 sorted nodes: 0 2 3 4 path length: 19 ========== ========== Error code: 0 ==========
DIJKSTRA	===== DIJKSTRA ===== shortest path: 0 1 3 sorted nodes: 0 1 3 path length: 12 ========== ========== Error code: 0 ==========
DIJKSTRAMIN	====== DIJKSTRAMIN ====== shortest path: 0 1 3 sorted nodes: 0 1 3 path length: 12 ========== ========== Error code: 0 ==========
BELLMANFORD	===== BELLMANFORD ===== shortest path: 1 2 4 sorted nodes: 1 2 4 path length: 9 ========== ========== Error code: 0 ==========
FLOYD	===== FLOYD ===== 0 6 11 12 15 7 0 5 6 9 2 8 0 6 4 10 6 8 0 3

	7 8 5 2 0 ========== ========== Error code: 0 ==========

표 4. 출력 형식

□ 동작 예시

command.txt	mapdata.txt
LOAD mapdata.txt PRINT DFS 0 3 DIJKSTRA 0 3 DFS 1 4 BELLMANFORD 1 4 DIJKSTRA -1 10 BELLMANFORD ASTAR 1 4 FLOYD	5 0 6 13 0 0 0 0 5 6 0 2 0 0 7 4 0 6 0 0 3 0 0 5 2 0

실행 결과 화면

log.txt

```
===== LOAD =====
Success
==========

==========
Error code: 0
==========

===== PRINT =====
0 6 13 0 0
0 0 5 6 0
2 0 0 7 4
0 6 0 0 3
0 0 5 2 0
==========

==========
Error code: 0
```

```
=================

===== DFS =====
shortest path: 0 2 4 3
sorted nodes: 0 2 3 4
path length: 19
================

=================
Error code: 0
=================

===== DIJKSTRA =====
shortest path: 0 1 3
sorted nodes: 0 1 3
path length: 12
================

=================
Error code: 0
=================

===== DFS =====
shortest path: 1 3 4
sorted nodes: 1 3 4
path length: 9
================

=================
Error code: 0
=================

===== BELLMANFORD =====
shortest path: 1 2 4
sorted nodes: 1 2 4
path length: 9
================

=================
Error code: 0
=================

===== DIJKSTRA =====
```

```
InvalidVertexKey
=================

=================
Error code: 201
=================

===== BELLMANFORD =====
VertexKeyNotExist
=================

=================
Error code: 200
=================

===== ASTAR =====
NonDefinedCommand
=================

=================
Error code: 300
=================

===== FLOYD =====
0 6 11 12 15
7 0 5 6 9
2 8 0 6 4
10 6 8 0 3
7 8 5 2 0
================

=================
Error code: 0
=================
```

표 5. 동작 예시

□ 구현 시 반드시 정의해야하는 Class의 멤버 변수

1. Manager: 다른 클래스들의 동작을 관리하여 프로그램을 전체적으로 조정하는 역할을 수행

항목	내용	비고
const char* RESULT_LOG_PATH	결과 파일 이름(log.txt)	값 수정 금지
std::ofstream fout	결과 파일의 FileStream	
Graph m_graph	그래프 클래스	

2. Graph

항목	내용	비고
Vertex* m_pVHead	Vertex Linked-List의 Head 포인터	
int m_vSize	Vertex의 수	

3. Edge

항목	내용	비고
int m_key	Edge(End Vertex)의 Key	
int m_weight	Edge의 Weight	
Edge* m_pNext	다음 Edge를 가리키는 포인터	

4. Vertex

항목	내용	비고
int m_key	Vertex(Start Vertex)의 Key	
int m_size	연결된 Edge의 수	
Edge* m_pEHead	Edge Linked-List의 Head 포인터	
Vertex* m_pNext	다음 Vertex를 가리키는 포인터	

□ Files

- ✓ command.txt : 프로그램을 동작시키는 명령어들을 저장하고 있는 파일
- ✓ mapdata.txt: 도로 정보 데이터를 저장하고 있는 파일
- ✓ log.txt: 모든 출력 결과를 저장하고 있는 파일 (파일 이름 변경 금지)

□ 제한사항 및 구현 시 유의사항

- ✓ 반드시 제공되는 압축파일(project3.tar.gz)을 이용하여 구현하며, 작성된 소스 코드의 이름과 클래스, 함수 이름 및 형태를 임의로 변경하지 않는다.
- ✓ 제공된 클래스의 함수 및 변수는 자유롭게 추가 구현이 가능하다.
  - 제공된 코드의 변수 및 함수 프로토타입은 변경 불가능
  - 함수 및 변수 추가가 가능하되, 제공된 코드의 함수로 테스트하여 채점하기 때문에 제공된 코드의 함수 기능이 제대로 구현되어야 한다.
  - MinHeap 클래스의 Get, DecKey 함수는 구현 방법에 따라 동작하지 않아도 무관하다.
- ✓ 메인 함수(main.cpp)는 절대 변경하지 않는다.

✓ 제시된 클래스를 각 기능에 알맞게 모두 사용한다.
✓ 프로그램 구조에 대한 디자인이 최대한 간결하도록 고려하여 설계한다.
✓ 채점 시 코드를 수정해야하는 일이 없도록 한다.
✓ 주석은 반드시 영어로 작성한다. (한글로 작성하거나 없으면 감점)
✓ 프로그램은 반드시 리눅스에서 동작해야한다. (컴파일 에러 발생 시 감점)
  - 제공되는 Makefile을 사용하여 테스트하도록 한다.
  - g++ 컴파일러의 버전은 4.8.4로 진행한다. (버전 확인 명령어: ~$ g++ --version)

□ 제출기한 및 제출방법

✓ 제출기한
  - 2018년 12월 12일 수요일 23:59:59 까지 제출
✓ 제출방법
  - 소스코드(Makefile과 텍스트 파일 제외)와 보고서 파일(pdf)을 함께 압축하여 U-Campus에 제출
    • 확장자가 .cpp, .h, .pdf가 아닌 파일은 제출하지 않음
    • 보고서 파일 확장자가 pdf가 아닐 시 감점
✓ 제출형식
  - 2000722000_DS_project3.tar.gz

✓ 압축방법
  - 압축하기

```
ex) /home/user/pr1 에 있는 모든 파일을 2000722000_DS_project3.tar.gz로 압축
  $ tar -zcvf 2000722000_DS_project3.tar.gz /home/user/pr1/*
```

  - 압축풀기

```
ex) project3.tar.gz의 압축을 해제
  $ tar -zxvf project3.tar.gz
```

✓ 보고서 작성 형식
  - 하드카피는 제출하지 않음
  - 보고서는 한글로 작성
  - 보고서에는 소스코드를 포함하지 않음
  - 반드시 포함해야하는 항목
    • Introduction : 프로젝트 내용에 대한 설명
    • Flowchart : 설계한 프로젝트의 플로우차트를 그리고 설명 (모든 명령어에 대하여 각각 그릴 것)
    • Algorithm : 프로젝트에서 사용한 알고리즘의 동작을 설명 (DFS, Dijkstra, Bellman-Ford, FLOYD에 대하여 각각 예시를 들어 설명할 것)
    • Result Screen : 모든 명령어에 대해 결과화면을 캡처하고 동작을 설명
    • Consideration : 고찰 작성

※ 프로젝트 관련 질문은 질문 게시판에 올려주세요. (http://datasci.kw.ac.kr:2018/)
메일로 질문하시면 답변해 드리지 않습니다.
3차 프로젝트 질문은 (12/12) 오후 5시까지만 받겠습니다.