Data Structure Lab. Project #3

2024년 11월 18일

Due date: 2024년 12월 14일 토요일 23:59:59 까지

본 프로젝트에서는 그래프를 이용해 그래프 연산 프로그램을 구현한다. 이 프로그램은 그래프 정보가 저장된 텍스트 파일을 통해 그래프를 구현하고, 그래프의 특성에 따라 BFS, DFS, Kruskal, Dijkstra, Bellman-Ford 그리고 FLOYD 연산을 수행한다. 그래프 데이터는 방향성(Direction)과 가중치(Weight)를 모두 가지고 있으며, 데이터 형태에 따라 List 그래프와 Matrix 그래프로 저장한다. BFS와 DFS는 그래프의 방향성과 가중치를 고려하지 않고, 그래프 순회 또는 탐색 방법을 수행한다. Kruskal 알고리즘은 최소 비용 신장 트리(MST)를 만드는 방법으로 방향성이 없고, 가중치가 있는 그래프 환경에서 수행한다. Dijkstra 알고리즘은 정점 하나를 출발점으로 두고 다른 모든 정점을 도착점으로 하는 최단경로 알고리즘으로 방향성과 가중치 모두 존재하는 그래프 환경에서 연산을 수행한다. 만약 weight가 음수일 경우 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 출력 파일(log.txt)에 저장한다.

각 그래프 연산들은 'GraphMethod' 헤더 파일에 일반 함수로 존재하며, 그래프 형식(List, Matrix)에 상관없이 그래프 데이터를 입력받으면 동일한 동작을 수행하도록 일반화하여 구현한다. 또한 충분히 큰 그래프에서도 모든 연산을 정상적으로 수행할 수 있도록 구현한다.

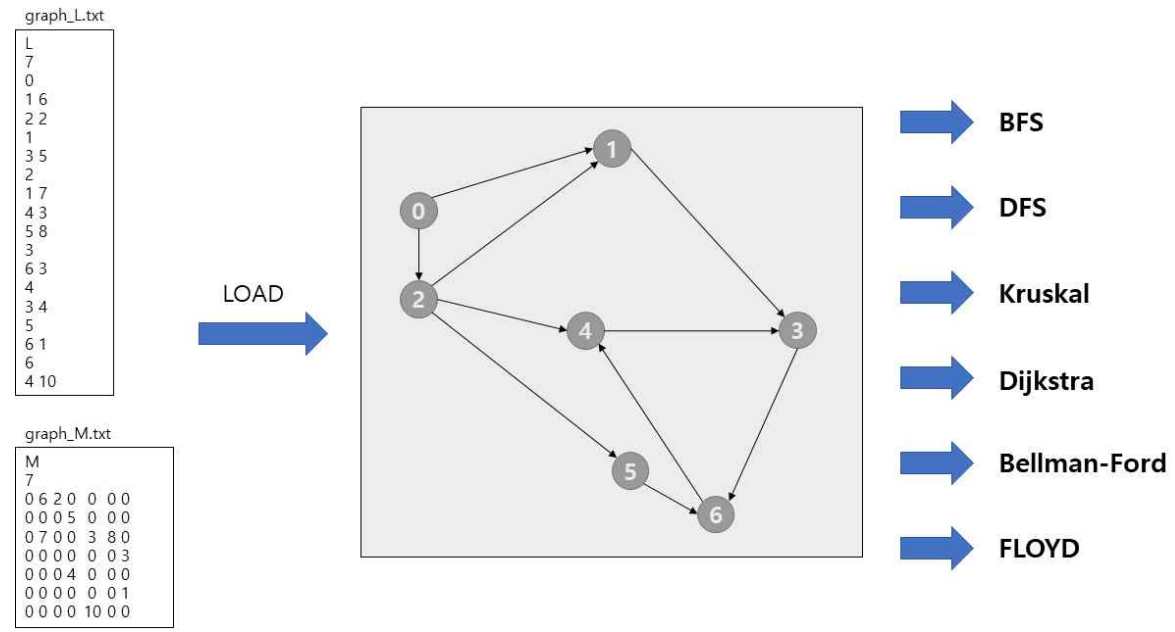

그림 1. 프로그램 구성

□ Program implementation

1. 그래프 데이터

프로그램은 그래프 정보가 저장되어 있는 텍스트 파일(graph_M.txt 또는 graph_L.txt)을 읽어, 해당 정보를 Adjacency List 또는 Adjacency Matrix에 저장한다. 텍스트 파일의 첫 번째 줄에는 그래프의 형식 정보(L: List, M: Matrix)가 저장되어 있고 두 번째 줄에는 그래프 크기가 저장되어 있다. 이후 데이터는 그래프 형식에 따라 구분된다. List그래프의 경우 표1-1과 같이 edge의 출발 vertex를 의미하는 데이터(형식-1), edge의 도착 vertex와 weight가 쌍으로 이루어진 데이터(형식-2)로 구성된다. 만약 edge가 없는 vertex의 경우 (형식-1)만 존재할 수 있다. Matrix그래프의 경우 표1-2의 (형식-3)과 같이 그래프의 정보가 저장되어 있다. 행렬의 행과 열은 각각 to vertex와 from vertex를 의미하고, 행렬의 값은 from vertex와 to vertex 사이를 잇는 edge의 weight를 의미한다. 이때, 그래프의 모든 vertex의 값은 0 이상의 정수이며, 0부터 시작하여 1, 2, 3, 4... 순으로 정해진다고 가정한다.

텍스트 파일에 포함되어 있는 그래프의 형식 및 그래프 크기 정보는 'Graph' 클래스에 저장하며, 그래프 연결 정보는 그래프 형식에 따라 알맞은 '그래프 클래스'에 저장한다. Adjacency List 정보는 'ListGraph' 클래스에 저장하고 Adjacency Matrix 정보는 'MatrixGraph' 클래스에 저장한다. 그래프 클래스들은 'Graph' 클래스를 상속받는다. 그래프 정보가 포함되어 있는 텍스트 파일의 형식은 그림 2-1과 그림 2-2와 같다. 각 클래스의 멤버 변수에 대한 자세한 내용은 '구현 시 반드시 정의해야하는 Class 및 멤버 변수'에 작성되어 있다.

형식	내용
1	시작 vertex
2	[도착 vertex] [weight]

표 1-1. List 그래프 데이터 형식

형식	내용
3	[weight_1] [weight_2] [weight_3] ... [weight_n]

표 1-2. Matrix 그래프 데이터 형식

```
L
7
0
1 6
2 2
1
3 5
2
1 7
4 3
5 8
3
6 3
4
3 4
5
6 1
6
4 10
```

그림 2-1. List 그래프의 정보가 저장되어 있는 텍스트 파일의 예

```
M
7
0 6 2 0  0  0 0
0 0 0 5  0  0 0
0 7 0 0  3  8 0
0 0 0 0  0  0 3
0 0 0 4  0  0 0
0 0 0 0  0  0 1
0 0 0 0 10 0 0
```

그림 2-2. Matrix 그래프의 정보가 저장되어 있는 텍스트 파일의 예

2. 그래프 연산

프로그램은 텍스트 파일로부터 그래프 정보를 읽은 뒤, 명령어에 따라 그래프 특성에 맞는 그래프 연산을 수행할 수 있다.

- BFS, DFS, DFS_R

각 명령어는 방향성과 가중치가 없는 그래프에서 수행 가능하므로 방향성과 그래프의 가중치가 없다고 가정하고 수행한다. 프로그램은 각 명령어와 함께 입력받은 vertex를 시작으로 각 명령어에 맞는 알고리즘을 이용하여 모든 vertex들을 방문하고, vertex의 방문 순서를 명령어의 결과로 출력한다. 이때, 각 명령어는 vertex의 값이 작은 vertex를 먼저 방문한다고 가정한다. BFS 명령어는 Queue를, DFS 명령어는 Stack을 이용하여 동작하고 DFS_R 명령어의 경우 함수의 재귀적 호출 방법을 사용하여 동작한다.

- KRUSKAL

Kruskal 명령어는 방향성이 없고 가중치가 있는 그래프에서 수행 가능하다. 프로그램은 Kruskal 명령어 수행 시, 저장되어 있는 그래프 연결 정보를 이용하여 해당 그래프에 대한 MST를 구하는 동작을 수행한다. 이때, MST를 구하는 과정에 있어서 sub-tree가 cycle을 이루는지에 대한 검사를 수행하도록 한다.

- DIJKSTRA

Dijkstra 명령어는 방향성과 가중치가 있는 그래프에서 수행 가능하다. 프로그램은 명령어와 함께 입력받은 vertex를 기준으로 Dijkstra 알고리즘을 수행한 후, 모든 vertex에 대해 shortest path를 구하고, shortest path에 대한 cost 값을 구한다.

- BELLMANFORD

Bellmanford 명령어는 Start Vertex와 End Vertex를 입력으로 받아 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 Kruskal 알고리즘의 edge를 정렬하는 데 사용한다. 연산의 효율을 향상하기 위해 segment size에 따라 정렬 알고리즘을 구현한다. 해당 프로젝트에서는 퀵 정렬을 수행하며 정렬 수행 시 segment size를 재귀적으로 분할 할 때, segment size가 6 이하일 경우 삽입정렬을 수행하며, segment size가 7 이상인 경우엔 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_size)
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 3. STL sort 의사코드

□ Functional Requirements

프로그램은 명령어를 통해 동작하며, 실행할 명령어가 작성되어 있는 텍스트 파일(command.txt)를 읽고 명령어에 따라 순차적으로 동작한다. 모든 명령어는 반드시 대문자로 입력하며, 각 명령어의 사용법과 기능은 다음과 같다.

명령어	명령어 사용 예 및 기능
LOAD	사용법) LOAD [textfile] 예시) LOAD graph_L.txt ✓ 프로그램에 그래프 정보를 불러오는 명령어로, 그래프 정보가 저장되어 있는 텍스트 파일(graph_L.txt 또는 graph_M.txt)을 읽어 그래프를 구성한다. ✓ 텍스트 파일이 존재하지 않거나, 비어있는 경우, 출력 파일(log.txt)에 오류 코드(100)를 출력한다. ✓ 기존에 그래프 정보가 존재할 때 LOAD가 입력되면 기존 그래프 정보는 삭제하고 새로 그래프를 생성한다.
PRINT	사용법) PRINT ✓ 그래프의 상태를 출력하는 명령어로, List형 그래프일 경우 adjacency list를, Matrix형 그래프일 경우 adjacency matrix를 출력한다. ✓ vertex는 오름차순으로 출력한다. edge는 vertex를 기준으로 오름차순으로 출력한다. ✓ 만약 그래프 정보가 존재하지 않을 경우, 출력 파일(log.txt)에 알맞은 오류 코드를 출력한다.

BFS	사용법) BFS [vertex] 예시) BFS 0 ✓ 현재 그래프에서 입력한 vertex를 기준으로 BFS를 수행하는 명령어로, BFS 결과를 출력 파일(log.txt)에 출력한다. ✓ BFS 명령어를 수행하면서 방문하는 vertex의 순서로 결과를 출력한다. ✓ 입력한 vertex가 그래프에 존재하지 않거나, vertex를 입력하지 않은 경우, 출력 파일(log.txt)에 알맞은 오류 코드를 출력한다.
DFS	사용법) DFS [vertex] 예시) DFS 0 ✓ 현재 그래프에서 입력한 vertex를 기준으로 DFS를 수행하는 명령어로, DFS 결과를 출력 파일(log.txt)에 출력한다. ✓ DFS 명령어를 수행하면서 방문하는 vertex의 순서로 결과를 출력한다. ✓ 입력한 vertex가 그래프에 존재하지 않거나, vertex를 입력하지 않은 경우, 출력 파일(log.txt)에 알맞은 오류 코드를 출력한다.
DFS_R	사용법) DFS_R [vertex] 예시) DFS_R 2 ✓ 현재 그래프에서 입력한 vertex를 기준으로 DFS_R 수행하는 명령어로, DFS_R 결과를 출력 파일(log.txt)에 출력한다. ✓ DFS_R 명령어를 수행하면서 방문하는 vertex의 순서로 결과를 출력한다. ✓ 입력한 vertex가 그래프에 존재하지 않거나, vertex를 입력하지 않은 경우, 출력 파일(log.txt)에 알맞은 오류 코드를 출력한다.
KRUSKAL	사용법) KRUSKAL ✓ 현재 그래프의 MST를 구하고, MST를 구성하는 edge들의 weight 값을 오름차순으로 출력하고 weight의 총합을 출력 파일(log.txt)에 출력한다. ✓ 입력한 그래프가 MST를 구할 수 없는 경우, 명령어를 수행할 수 없는 경우, 출력 파일(log.txt)에 알맞은 오류 코드를 출력한다.
DIJKSTRA	사용법) DIJKSTRA [vertex] 예시) DIJKSTRA 1 ✓ 명령어의 결과인 shortest path와 cost를 출력 파일(log.txt)에 출력한다. 출력 시 vertex, shortest path, cost 순서로 출력하며, shortest path는 해당 vertex에서 기준 vertex까지의 경로를 역순으로 출력한다. ✓ 기준 vertex에서 도달할 수 없는 vertex의 경우 'x'를 출력한다. ✓ 입력한 vertex가 그래프에 존재하지 않거나, vertex를 입력하지 않은 경우, 명령어를 수행할 수 없는 경우, 출력 파일(log.txt)에 알맞은 오류 코드를 출력한다.

BELLMANFORD	사용법) BELLMANFORD [vertex(start)] [vertex(end)] 예시) BELLMANFORD 0 3 ✓ StartVertex를 기준으로 Bellman-Ford를 수행하여 EndVertex까지의 최단 경로와 거리를 구하는 명령어로, Bellman-Ford 결과를 출력 파일(log.txt)에 저장한다. ✓ 음수인 weight가 있는 경우에도 동작해야한다. ✓ StratVertex에서 EndVertex로 도달할 수 없는 경우 'x'를 출력한다. ✓ 입력한 vertex가 그래프에 존재하지 않거나, 입력한 vertex가 부족한 경우, 명령어를 수행할 수 없는 경우, 음수 사이클이 발생한 경우 출력 파일(log.txt)에 알맞은 오류 코드를 출력한다.
FLOYD	사용법) FLOYD ✓ 모든 vertex의 쌍에 대해서 StartVertex에서 EndVertex로 가는데 필요한 비용의 최솟값을 행렬 형태로 출력한다. ✓ 기준 vertex에서 도달할 수 없는 vertex의 경우 'x'를 출력한다. ✓ 명령어를 수행할 수 없는 경우, 음수 사이클이 발생한 경우 출력 파일(log.txt)에 알맞은 오류 코드를 출력한다.
EXIT	사용법) EXIT ✓ 프로그램 상의 메모리를 해제하며, 프로그램을 종료한다.

표2. 명령어 기능 및 사용 예

□ Requirements in implementation

- ✓ 모든 명령어는 command.txt에 저장하여 순차적으로 읽고 처리한다.
- ✓ 모든 명령어는 반드시 대문자로 입력하며, vertex는 숫자로 입력한다.
- ✓ 명령어에 인자(Parameter)가 모자라거나 필요 이상으로 입력받을 경우 오류 코드를 출력한다.
- ✓ 예외처리에 대해 반드시 오류 코드를 출력한다.
- ✓ 출력은 "출력 포맷"을 반드시 따라한다.
- ✓ log.txt 파일에 출력 결과를 반드시 저장한다.
  - log.txt가 이미 존재할 경우 텍스트 파일 가장 뒤에 이어서 추가로 저장한다.

□ Error Code

명령어에 인자가 모자라거나 필요 이상으로 입력 받을 경우, 또는 각 명령어 마다 오류 코드를 출력해야 하는 경우, 출력 파일(log.txt)에 명령어 별로 알맞은 오류 코드를 출력한다. 출력 파일(log.txt)이 이미 존재할 경우, 기존 내용 뒤에 이어서 추가로 출력하여 저장한다. 각 명령어별 오류 코드는 다음과 같다.

동작	오류 코드
LOAD	100
PRINT	200
BFS	300
DFS	400
DFS_R	500
KRUSKAL	600
DIJKSTRA	700
BELLMANFORD	800
FLOYD	900

표3. 에러 코드

□ Print Format

명령어 동작이 성공하였거나 실패한 경우, 출력 파일(log.txt)에 해당 내용을 지정된 형식에 맞추어 출력한다. 출력 파일(log.txt)이 이미 존재할 경우, 기존 내용 뒤에 이어서 추가로 저장한다. 명령어별 출력 형식은 다음과 같다.

<table>
<tr><th>기능</th><th colspan="2">출력 포맷</th></tr>
<tr><td>오류(ERROR)</td><td colspan="2">======== ERROR ========
100
======================</td></tr>
<tr><td>LOAD</td><td colspan="2">======== LOAD ========
Success
======================</td></tr>
<tr><td>PRINT</td><td>List그래프의 경우)
======== PRINT========
[0] -> (1,6) -> (2,2)
[1] -> (3,5)
[2] -> (1,7) -> (4,3) -> (5,8)
[3] -> (6,3)
[4] -> (3,4)
[5] -> (6,1)
[6] -> (4,10)
======================</td><td>Matrix그래프의 경우)
======== PRINT========
[0] [1] [2] [3] [4] [5] [6]
[0] 0 6 2 0 0 0 0
[1] 0 0 0 5 0 0 0
[2] 0 7 0 0 3 8 0
[3] 0 0 0 0 0 0 3
[4] 0 0 0 4 0 0 0
[5] 0 0 0 0 0 0 1
[6] 0 0 0 0 10 0 0
======================</td></tr>
<tr><td>BFS</td><td colspan="2">======== BFS ========
startvertex: 0
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6
======================</td></tr>
</table>

DFS	======== DFS ======== startvertex: 0 0 -> 1 -> 2 -> 4 -> 3 -> 6 -> 5 ======================
DFS_R	======== DFS_R======== startvertex: 2 2 -> 0 -> 1 -> 3 -> 4 -> 6 -> 5 ======================
Kruskal	====== Kruskal ======= [0] 2(2) [1] 3(5) [2] 0(2) 4(3) [3] 1(5) 4(4) 6(3) [4] 2(3) 3(4) [5] 6(1) [6] 3(3) 5(1) cost: 18 ======================
Dijkstra	====== Dijkstra ======= startvertex: 5 [0] x [1] x [2] x [3] 5 -> 6 -> 4 -> 3 (15) [4] 5 -> 6 -> 4 (11) [6] 5 -> 6 (1) ======================
Bellman-Ford	====== Bellman-Ford ======= 0 -> 2 -> 5 -> 6 cost: 11 ======================
FLOYD	======== FLOYD ======== [0] [1] [2] [3] [4] [5] [6] [0] 0 6 2 9 5 10 11 [1] x 0 0 5 18 x 8 [2] x 7 0 7 3 8 9 [3] x x x 0 13 x 3 [4] x x x 4 0 x 7 [5] x x x 15 11 0 1 [6] x x x 14 10 x 0 ======================

표4. 출력 형식

□ 동작 예시

command.txt

```
LOAD graph_M.txt
PRINT
BFS 3
DIJKSTRA 5
EXIT
```

실행 결과 화면 및 log.txt

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

========== PRINT ==========
    [0] [1] [2] [3] [4] [5] [6]
[0]  0   6   2   0   0   0   0
[1]  0   0   0   5   0   0   0
[2]  0   7   0   0   3   8   0
[3]  0   0   0   0   0   0   3
[4]  0   0   0   4   0   0   0
[5]  0   0   0   0   0   0   1
[6]  0   0   0   0   10  0   0
==========================

========== BFS ==========
3 -> 1 -> 4 -> 6 -> 0 -> 2 -> 5
==========================

========= Dijkstra =========
startvertex: 5
[0] x
[1] x
[2] x
[3] 5 -> 6 -> 4 -> 3 (15)
[4] 5 -> 6 -> 4 (11)
[6] 5 -> 6 (1)
==========================
```

표5. 동작 예시

□ 구현 시 반드시 정의해야 하는 Class와 헤더파일

1. Graph : Graph 클래스

항목	내용	비고
m_Type	그래프 형식	List일 경우 0, Matrix일 경우 1
m_Size	그래프 크기	

표6-1. Graph 클래스 멤버 변수

2. ListGraph : ListGraph 클래스

항목	내용	비고
m_List	그래프 데이터	map < int, int >* (map[from vertex].insert([to vertex], [weight]))

표6-2. ListGraph 클래스 멤버 변수

3. MatrixGraph : MatrixGraph클래스

항목	내용	비고
m_Mat	그래프 데이터	int** ([from vertex][to vertex] = weight)

표6-3. MatrixGraph 클래스 멤버 변수

4. GraphMethod: GraphMethod 헤더 파일
   - 그래프 연산을 수행하는 일반 함수들을 모아둔 헤더 파일

항목	내용	비고
BFS		BFS(Graph* graph, int vertex)
DFS		DFS(Graph* graph, int vertex)
DFS_R	DFS 재귀연산	DFS_R(Graph* graph, vector<bool>* visit, int vertex)
Kruskal		Kruskal(Graph* graph)
Dijkstra		Dijkstra(Graph* graph, int vertex)
Bellmanford		BellanFord(Graph* graph, int s_vertex, int e_vertex)
FLOYD		FLOYD(Graph* graph)

표6-4. 그래프 연산 일반 함수

5. Manager : Manager 클래스
   - 다른 클래스들의 동작을 관리하여 프로그램을 전체적으로 조정하는 역할을 수행

항목	내용	비고
graph	그래프 클래스	

표6-5. Manager 클래스 멤버 변수

□ Files

- ✓ command.txt : 프로그램을 동작시키는 명령어들을 저장하고 있는 파일
- ✓ graph_L.txt : graph 데이터를 adjacency list 형식으로 저장하고 있는 파일
- ✓ graph_M.txt : graph 데이터를 adjacency matrix 형식으로 저장하고 있는 파일
- ✓ log.txt : 프로그램 출력 결과를 모두 저장하고 있는 파일

□ 제한사항 및 구현 시 유의사항

- ✓ 반드시 제공되는 코드(GitHub 주소 참고)를 이용하여 구현한다.
- ✓ 클래스의 함수 및 변수는 자유롭게 추가 구현이 가능하다.
  (Union, Find, Check, Sort, Compare 등의 함수 필요시 추가로 구현)
- ✓ 제시된 Class를 각 기능에 알맞게 모두 사용한다.
- ✓ 프로그램 구조에 대한 디자인이 최대한 간결하도록 고려하여 설계한다.
- ✓ 채점 시 코드를 수정해야 하는 일이 없도록 한다.
- ✓ 주석은 반드시 영어로 작성한다. (한글로 작성하거나 없으면 감점)
- ✓ 프로그램은 반드시 리눅스(Ubuntu 18.04)에서 동작해야 한다. (컴파일 에러 발생 시 감점)
  - 제공되는 Makefile을 사용하여 테스트하도록 한다.
- ✓ 메모리 누수가 발생하지 않도록 프로그램을 작성한다. (메모리 누수 발생 시 감점)

□ 제출기한 및 제출방법

- ✓ 제출기한
  - **2024년 12월 14일 토요일 23:59:59 까지 제출**

- ✓ 제출방법
  - 소스코드(Makefile과 텍스트 파일 제외)와 보고서 파일(pdf)을 함께 압축하여 제출
    - 확장자가 .cpp, .h, .pdf가 아닌 파일은 제출하지 않음(.txt 파일 제출 X)
  - KLAS -> 과제 제출 -> tar.gz로 과제 제출

- ✓ 제출 형식
  - 학번_DS_project3.tar.gz (ex. 2021202001_DS_project3.tar.gz)

- ✓ 보고서 작성 형식 및 제출 방법
  - Introduction : 프로젝트 내용에 대한 설명
  - Flowchart : 설계한 프로젝트의 플로우차트를 그리고 설명
  - Algorithm : 프로젝트에서 사용한 알고리즘의 동작을 설명
  - Result Screen : 모든 명령어에 대해 결과화면을 캡처하고 동작을 설명
  - Consideration : 고찰 작성
    - 위의 각 항목을 모두 포함하여 작성
    - 보고서 내용은 한글로 작성
    - 보고서에는 소스코드를 포함하지 않음