

데이터 구조 실습 보고서

학 과: 컴퓨터정보공학

담당교수: 이기훈 교수님

학 번: 2019202075

성 명: 김효석

Introduction

해당 프로젝트는 FP-Growth와 B+Tree를 이용하여 상품 추천 프로그램을 구현하는 것에 목적을 둔다.

이 프로그램은 장바구니 데이터(market.txt)에서 같이 구매한 상품들을 받아 FP-Growth를 구축한다. 이 때 FP-Growth는 상품들의 연관성을 Tree 구조로 저장하는 FP-Tree와 상품별 빈도수 및 정보, 해당 상품과 연결된 FP-Tree의 상품 노드들을 관리하는 Header Table로 구성된다.

이 때 FP-Tree에 저장된 연관상품들은 Frequent Pattern이라 하고 이것들은 SAVE 명령어를 통해 result.txt에 저장된다.

해당 result.txt 파일로부터 BTLOAD명령어를 통해 빈도수를 기준으로 B+Tree에 저장된다.

B+Tree는 IndexNode와 DataNode로 구분되며 Index Node는 DataNode를 찾기 위한 노드이며, DataNode는 Frequent Pattern들의 정보가 저장되어 있는 Node이다.

본 프로젝트를 구현할 때의 주의점은 다음과 같다.

- 모든 명령어는 command.txt에 저장하여 순차적으로 읽고 처리한다.
 - 모든 명령어는 반드시 대문자로 입력한다.
 - 명령어에 인자(Parameter)가 모자라거나 필요 이상으로 입력 받으면 에러코드를 출력한다.
 - 예외처리에 대해 반드시 에러 코드를 출력한다.
 - 출력은 "출력 포맷"을 반드시 따라야 한다.
 - log.txt 파일에 출력 결과를 반드시 저장한다.
- Log.txt가 이미 존재할 경우 파일 가장 뒤에 이어서 출력(저장)한다.
- 읽어야 할 텍스트 파일이 존재하지 않으면 해당 텍스트 파일을 생성한 뒤 진행하도록 한다.

본 프로젝트를 구현할 때의 반드시 선언해야 하는 class는 다음과 같다.

- FPGrowth
- FPNode
- HeaderTable
- FrequentPatternNode
- BpTreeNode
- BpTreeIndexNode
- BpTreeDataNode
- BpTree
- Manager

위의 class는 각각 다음의 역할을 가지고 있다.

-FPGrowth: FPNode와 HeaderTable을 가지며 market.txt에서 정보를 읽어와 Table을 구축한 후 해

당 Table을 기준으로 FPTree를 구축한다.

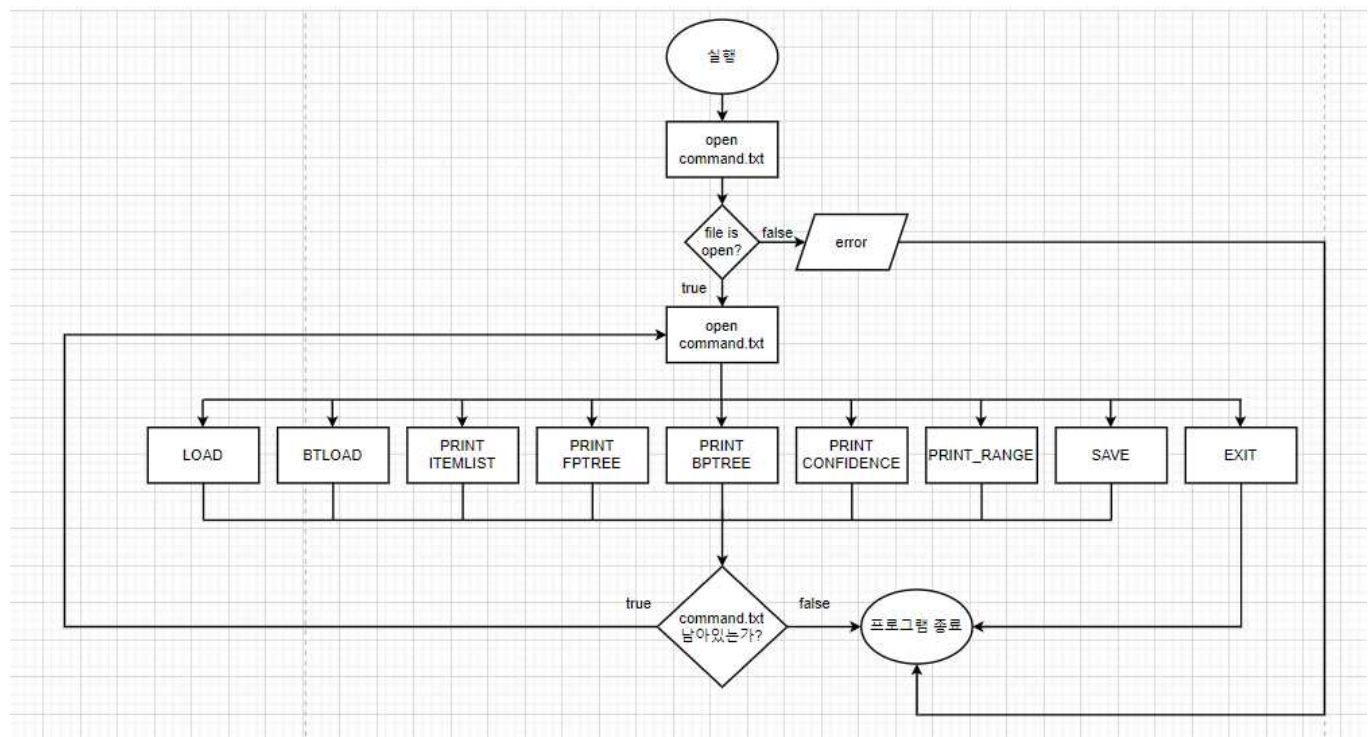
-BpTree: save 명령어를 통해 저장된 result.txt 파일을 읽어 frequency를 기준으로 BpTree를 구축한다.

-Manager: 다른 모든 클래스들의 동작을 관리하여 프로그램 전체적으로 조정하는 역할 수행

Flow chart

Manager

프로그램을 전반적으로 관리해주는 class이다. Manager class는 프로그램이 시작하면 command.txt로부터 명령어를 읽어와 해당하는 동작을 수행한다.

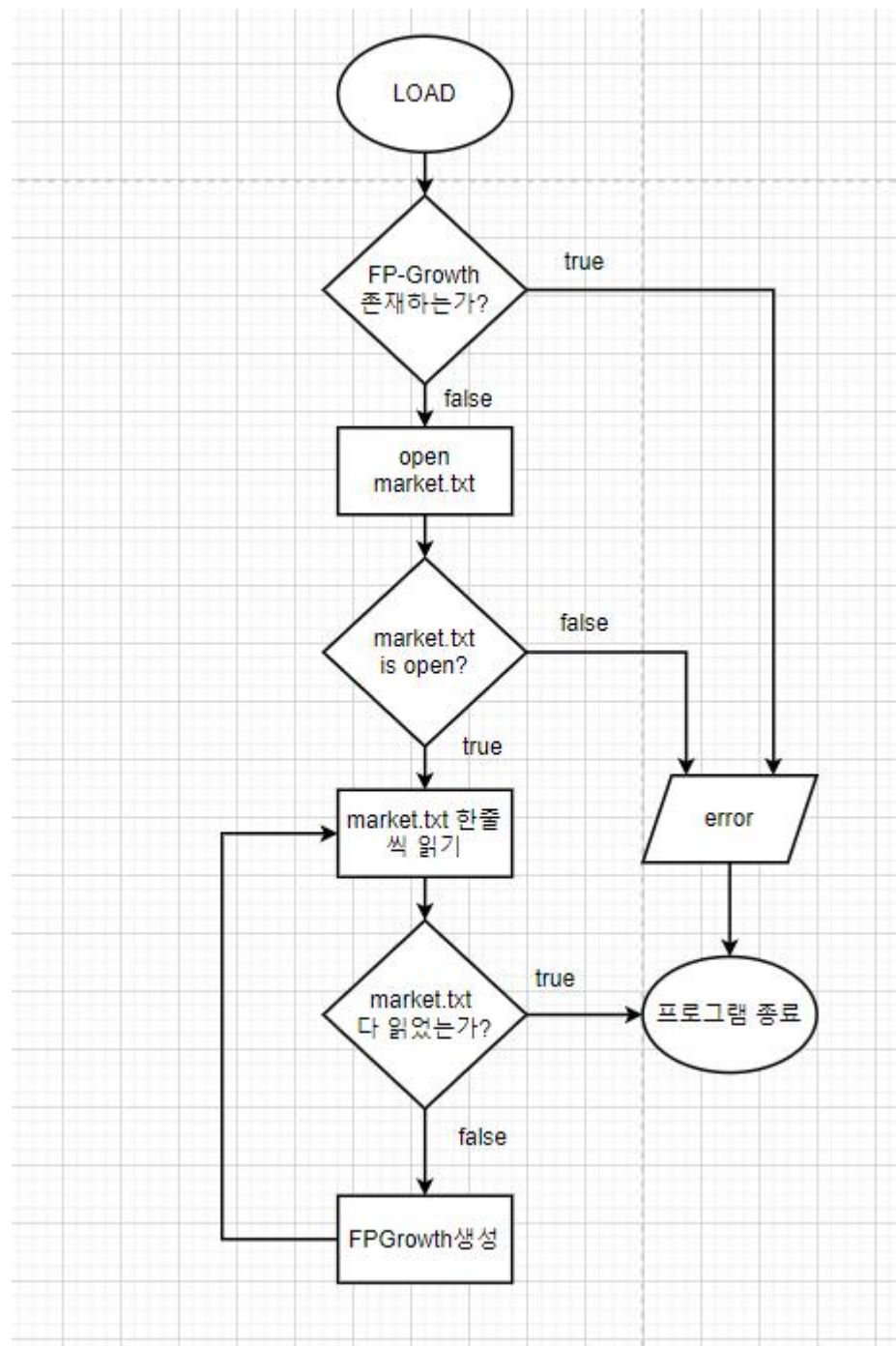


LOAD

LOAD 명령어는 "market.txt"에 저장되어 있는 데이터 정보를 불러오는 명령어이며, txt파일에 데이터 정보가 존재할 경우 해당 파일을 읽어 FP-Growth를 생성한다. 만약 텍스트 파일이 존재하지 않거나 자료구조가 이미 생성되어 있다면 에러코드를 출력한다.

<LOAD 조건>

- 상품명은 무조건 소문자이다.
- "Wt"를 구분자로 하여 상품을 구분한다.
- 같이 구매한 물품들은 줄 단위로 구분되어 있다.

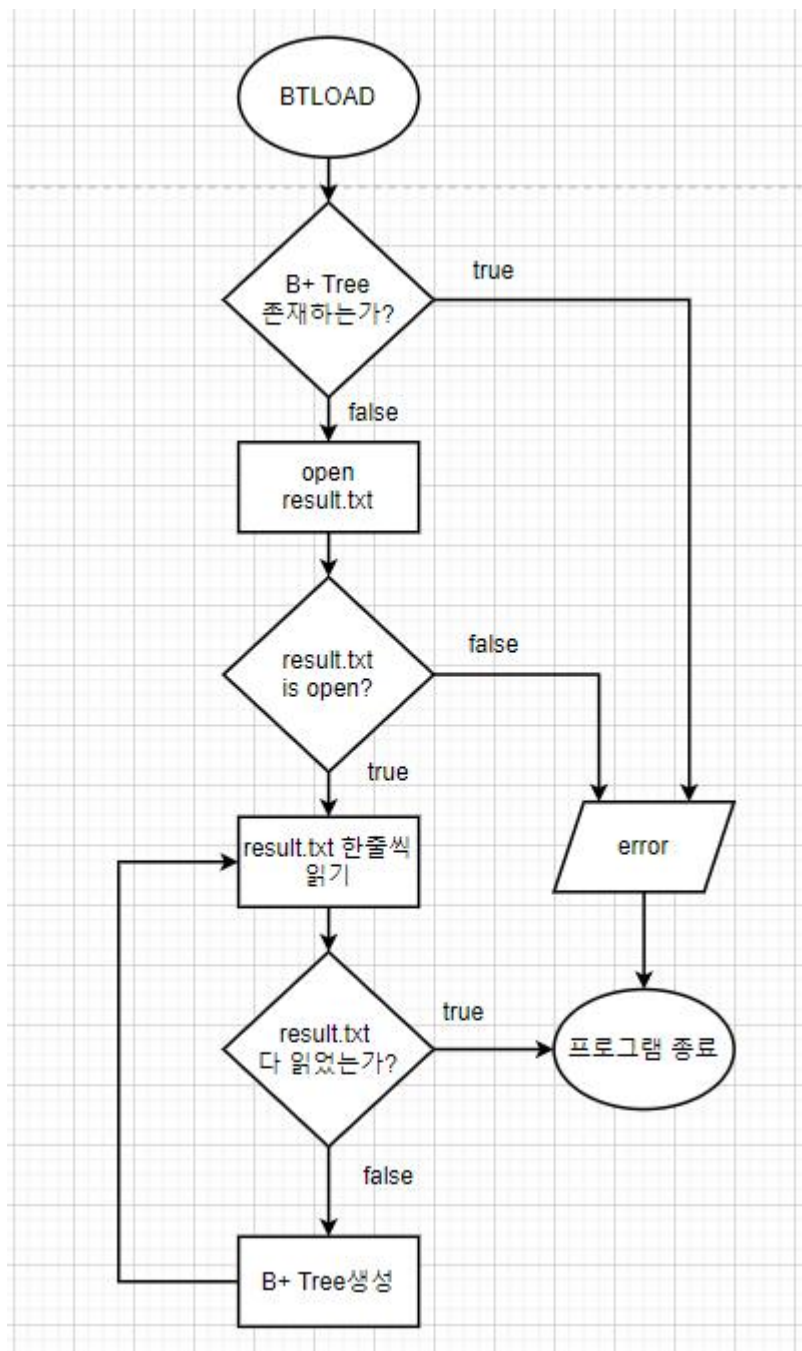


BTLOAD

BTLOAD 명령어는 result.txt 파일의 데이터 정보를 불러오는 명령어이며 정보가 존재하는 경우 B+-Tree에 저장한다. 만약 result.txt파일이 존재하지 않거나 B+-Tree가 이미 존재한다면 에러 코드를 출력한다.

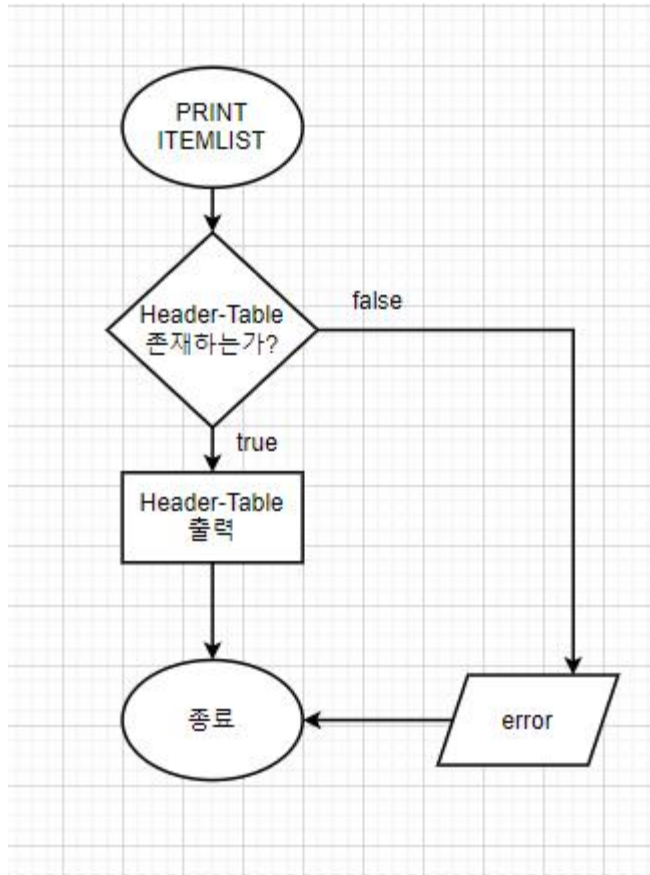
BTLOAD 조건

- Result.txt 파일에서 첫 번째 값은 빈도수이고, 이후 값은 상품 정보이다.
- "Wt"를 구분자로 하여 상품을 구분한다.
- Frequent Pattern에 속한 상품들은 줄 단위로 구분된다.



PRINT_ITEMLIST

PRINT_ITEMLIST 명령어는 FP-Growth의 Header Table에 저장된 상품들을 빈도수를 기준으로 내림차순으로 출력하는 명령어이다. 해당 명령어를 실행할 때 threshold보다 작은 빈도수를 가진 상품도 출력한다. 만약 Header Table이 비어 있다면 에러 코드 출력한다.



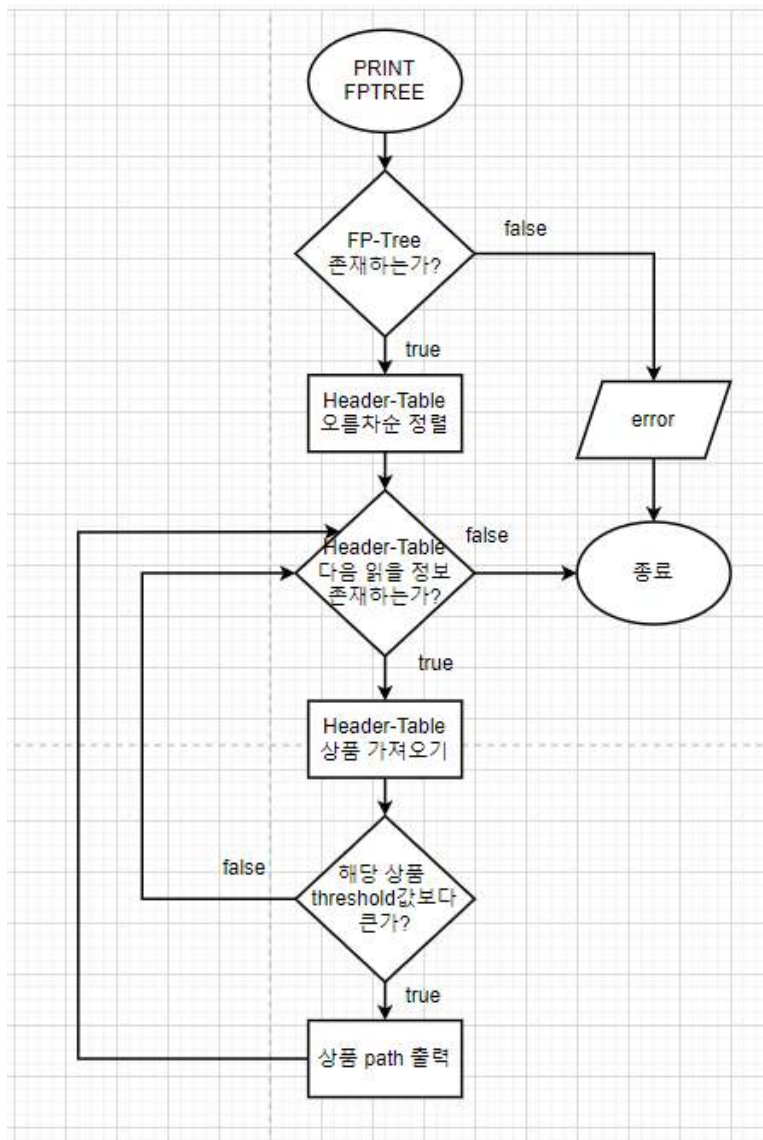
PRINT_FPTREE

PRINT_FPTREE 명령어는 FP-Growth의 FP-Tree 정보를 출력하는 명령어이다.

Header-Table을 빈도수를 기준으로 오름차순으로 정렬하고 threshold값 이상의 상품들을 출력한다. 만약 FP-Tree가 비어 있다면 에러 코드를 출력한다.

PRINT_FPTREE 명령어 출력 조건

- Header Table의 오름차순 순으로 FP-Tree의 path를 출력한다.
- threshold보다 작은 상품은 넘어간다.
- Header Table의 상품을 {상품명, 빈도수}로 출력한다.
- 해당 상품과 연결된 FP-Tree의 path들을 {상품명, 빈도수}로 root노드 전까지 연결된 부모 노드를 출력한다.
- 해당 상품과 연결된 다음 노드들이 없을 때까지 출력하며 다음 노드로 이동하면 다음 줄로 이동한다.



PRINT_BPTREE

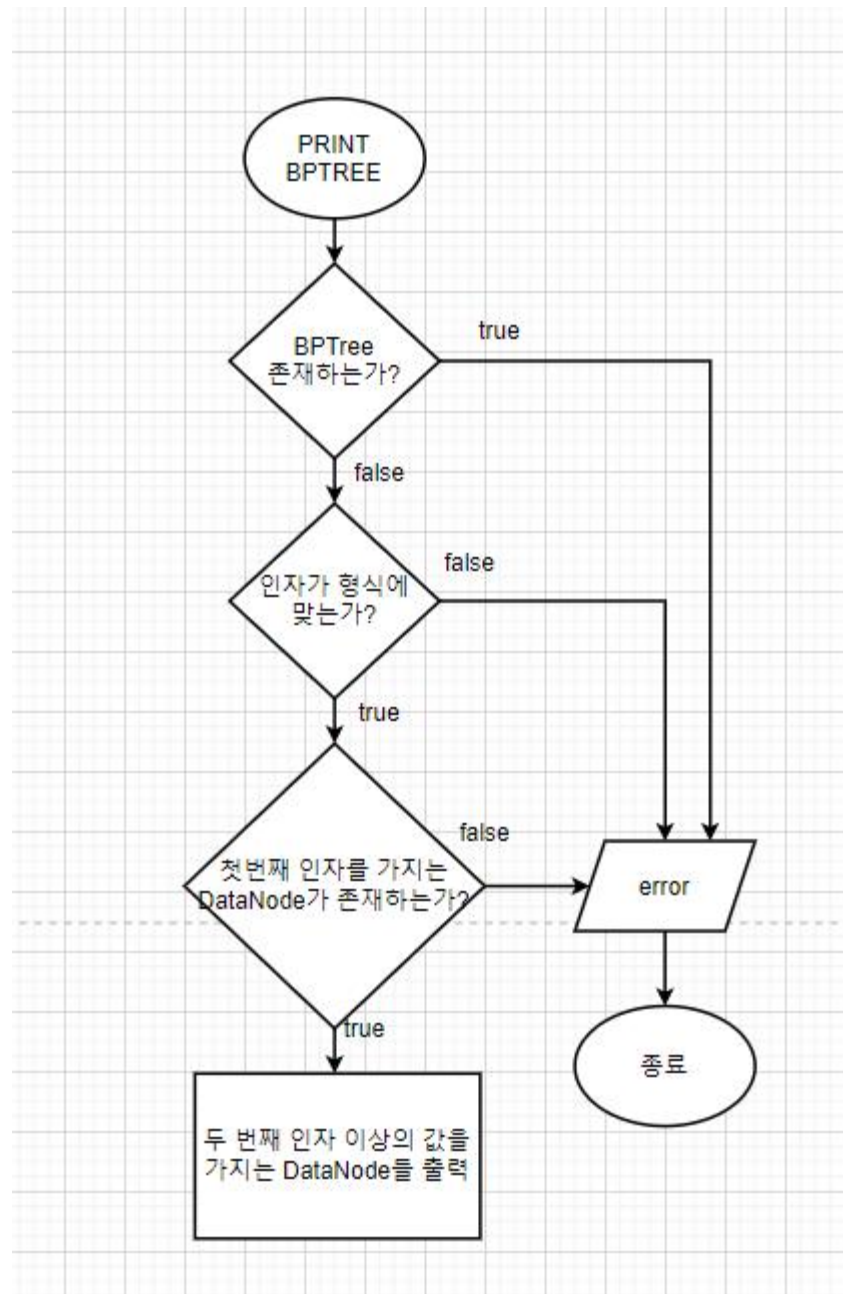
PRINT_BPTREE 명령어는 B⁺-Tree에 저장된 Frequent Pattern 중 입력된 상품과 최소 빈도수 이상의 값을 가지는 Frequent Pattern을 출력하는 명령어이다.

첫 번째 인자로 상품명을 입력 받고 두 번째 인자로 최소 빈도수를 입력 받는다.

최소 빈도수 이상의 입력된 상품을 포함하는 Frequent Pattern을 탐색하며 출력한다.

만약 인자가 부족하거나 형식이 다르다면 에러 코드를 출력한다.

또한 출력할 Frequent Pattern이 없거나 B⁺-Tree가 비어 있는 경우 에러 코드를 출력한다.



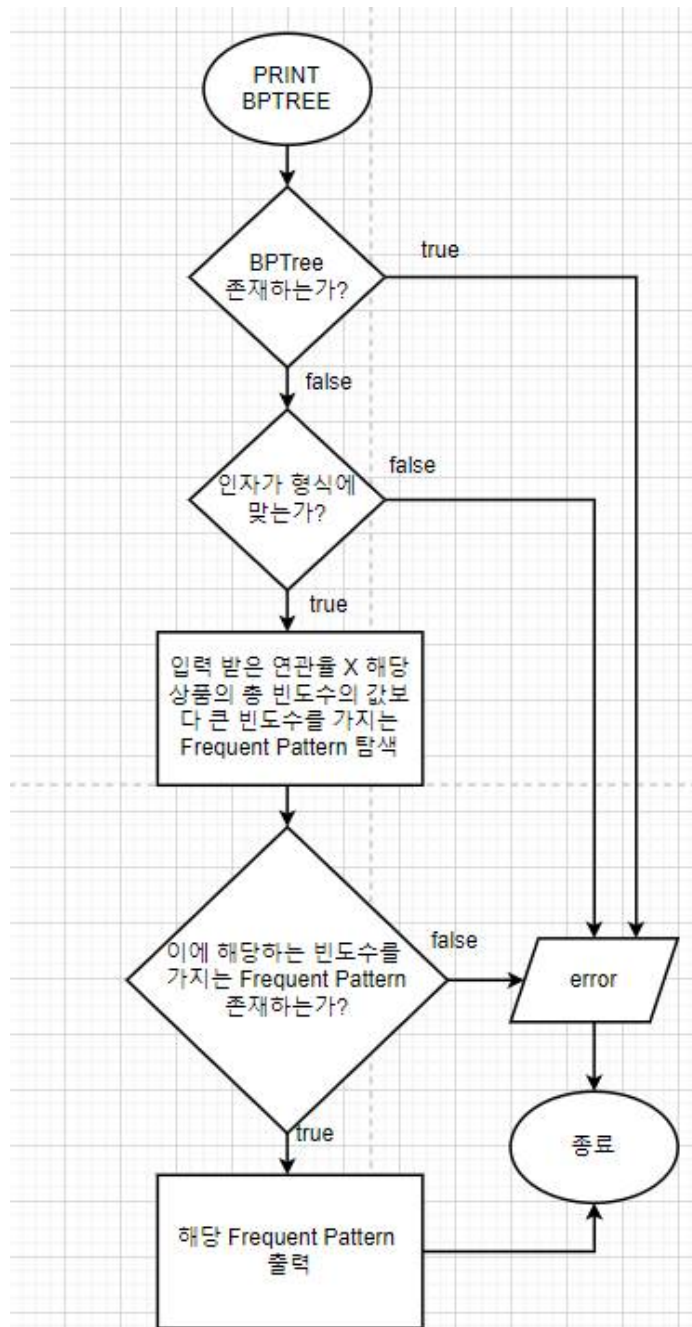
PRINT_CONFIDENCE

PRINT_CONFIDENCE 명령어는 B+-Tree에 저장된 Frequent Pattern 중 입력된 상품과 연관율 이상의 confidence 값을 가지는 Frequent Pattern을 출력하는 명령어이다.

첫 번째 인자로 상품명을 입력 받고 두 번째 인자로 연관율을 받는다.

연관율은 (부분집합의 빈도수)/(해당 상품의 총 빈도수)이다. 해당 명령이 실행되면 입력 받은 연관율과 해당 상품의 총 빈도수를 곱해서 해서 이 값보다 큰 빈도수를 가진 DataNode를 탐색한다. 만약 출력할 Frequent Pattern이 없거나 B+-Tree가 비어 있다면 에러 코드를 출력한다.

또한 2개의 인자가 모두 입력되지 않거나 형식이 다르다면 에러 코드를 출력한다.

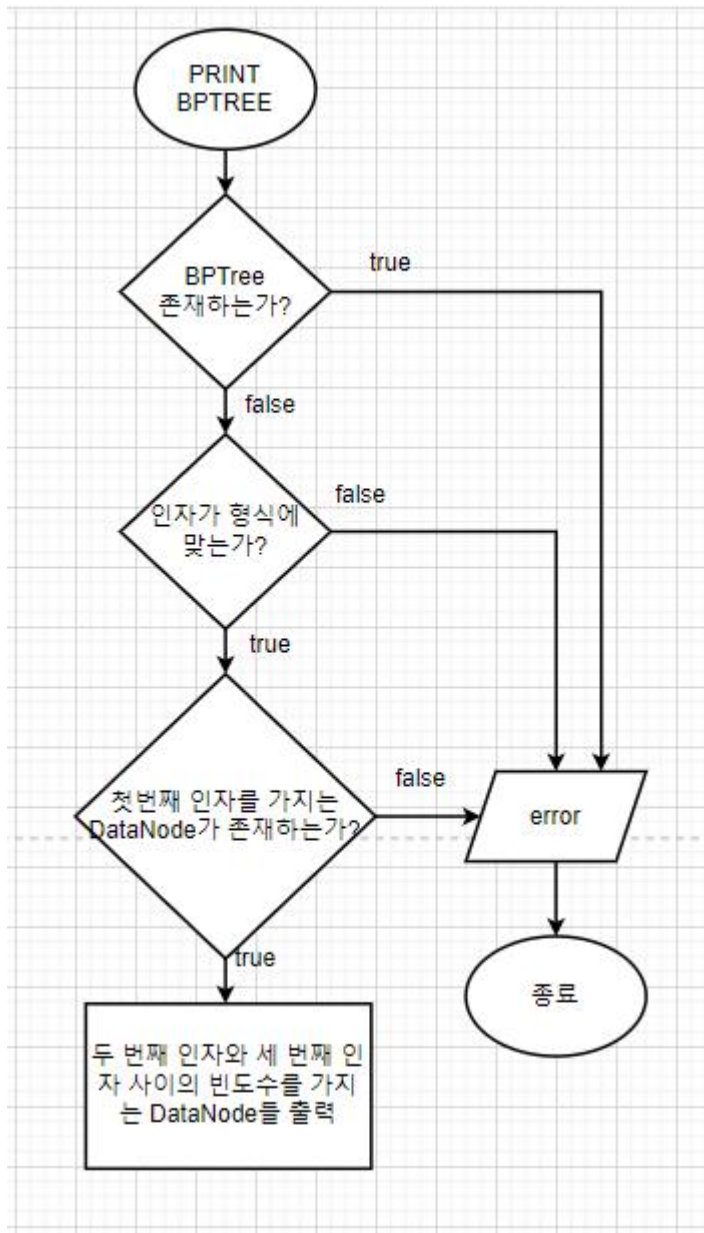


PRINT_RANGE

PRINT_RANGE 명령어는 B+-Tree에 저장된 Frequent Pattern을 출력하는 명령어이다.

PRINT_BPTREE의 명령어와 유사하며 다른 점은 인자를 하나 추가로 받아 두번째 인자와 세번째 인자 사이의 빈도수를 가지는 Frequent Pattern을 출력한다.

예러 코드는 PRINT_BPTREE와 같다.



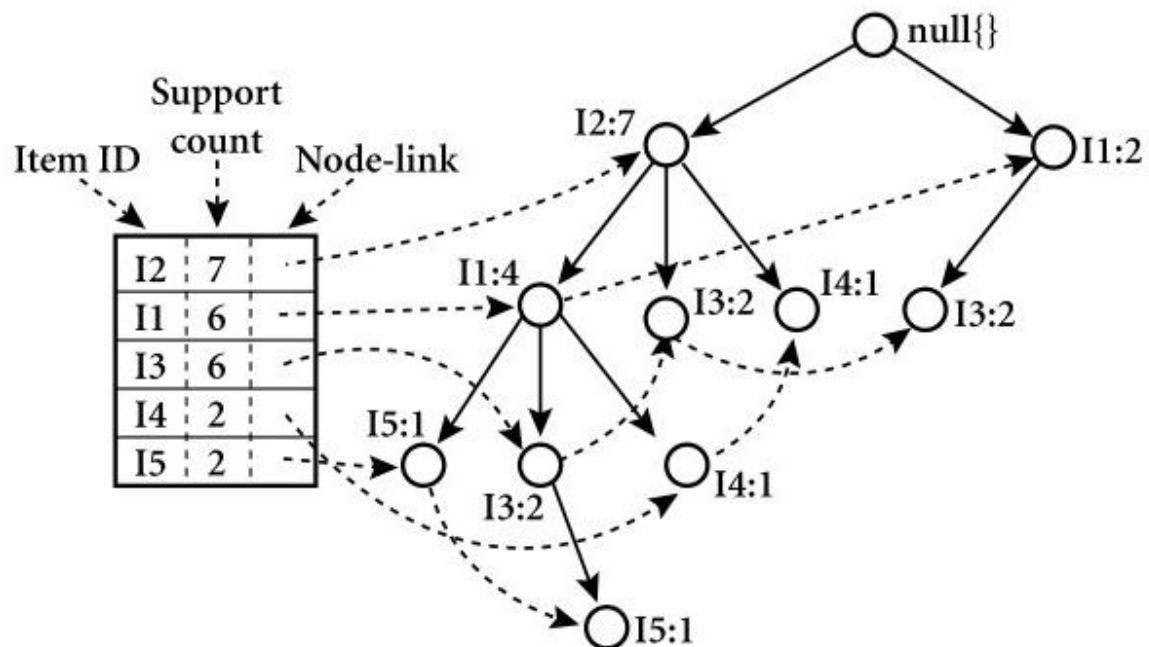
Algorithm

본 프로젝트에서 사용한 알고리즘은 FP-Growth, B⁺-Tree 총 2 개를 사용하였다.

<FP-Growth>

FP-Growth 알고리즘이란 Apriori 알고리즘을 개선한 알고리즘이다. Apriori 알고리즘에선 Pattern 을 찾기 위해 candidate 라는 길이에 따른 Pattern 후보를 만들고 해당 Pattern 의 support 를 구하기 위해 데이터를 스캔해야 한다. 즉, 데이터를 스캔하는 횟수가 적게는 1 번에서 많게는 가장 긴 트랜잭션의 아이템 개수까지 일어날 수 있다는 문제점을 가진다.

하지만 FP-Growth 알고리즘은 데이터를 읽는 횟수가 Apriori 알고리즘에 비해 적다. 그 이유는 Tree 와 Linked List 를 사용하기 때문이다.



위와 같은 그림이 나타내는 것은 FP-Growth 알고리즘을 도식화한것이다.

좌측의 Table 은 Node-Link 를 가진 배열(header Table)이다. 우측의 Node 들과 화살표는 Tree 이다. 이때 Tree 의 RootNode 는 nullptr 이다. 또한 Table 과 Tree 를 각각 화살표로 연결해 놓은 것은 Linked List 로 연결한다.

다음은 FPTree 의 동작 원리이다.

만약 데이터가 아래와 같이 존재한다고 가정하면,

Mineral water, milk, energy bar, green tea

fresh tuna, spaghetti, mineral water

spaghetti, milk, French fries

Spaghetti, mineral water, green tea

Mineral water, eggs, soup, French fries

각 item 의 전체 노출 수를 구한 후 빈도(support count)가 높은 순으로 정렬한다.

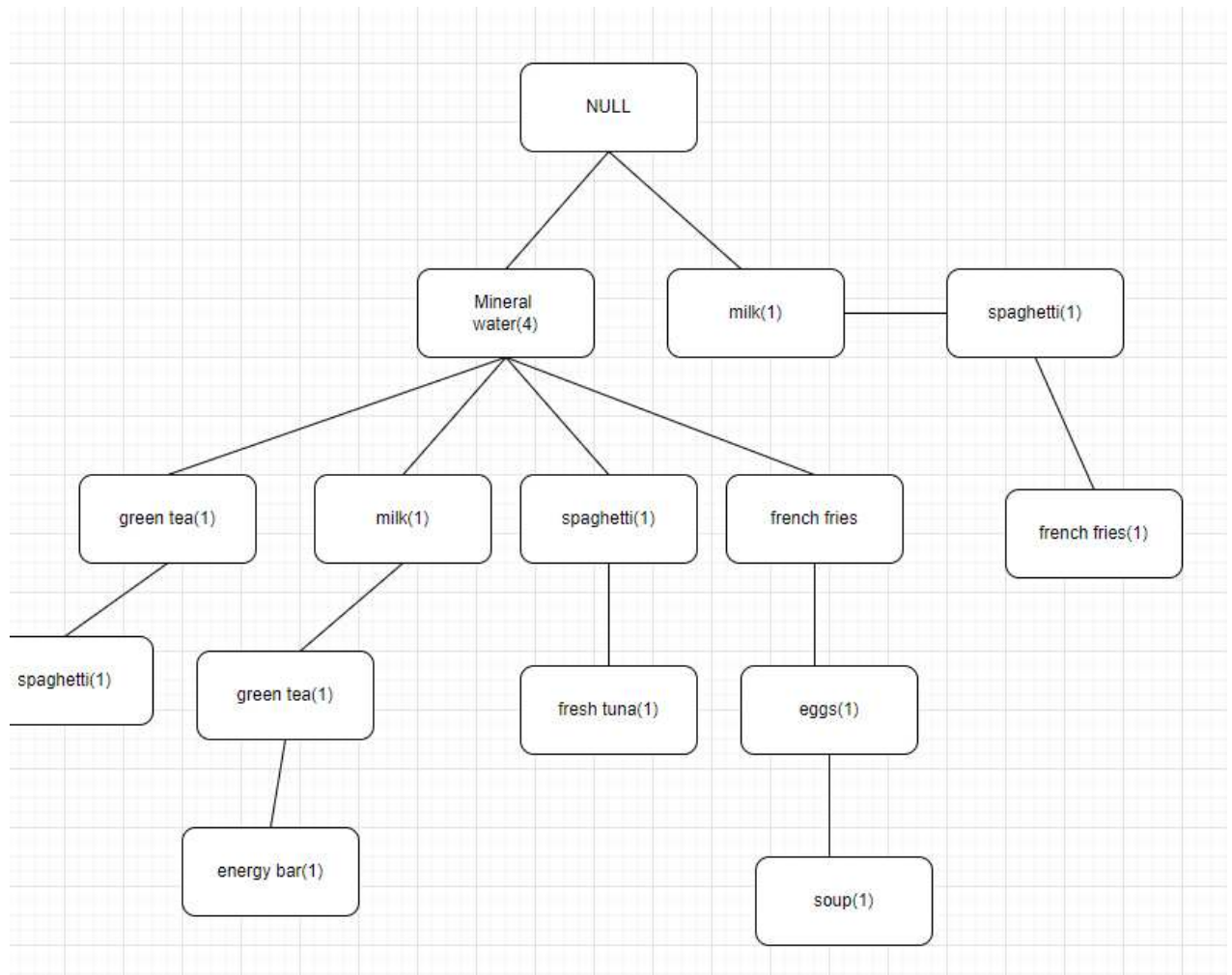
No.	Item	Support count
1	Mineral water	4
2	Milk	2
3	Green tea	2
4	Spaghetti	2
5	French fries	2
6	Fresh tuna	1
7	Eggs	1
8	Soup	1

위와 같이 support count 를 기준으로 정렬이 되었다면 이 순서를 기준으로 각각의 item 을 정렬한다.

1. Mineral water, milk, green tea, energy bar
2. Mineral water, spaghetti, fresh tuna
3. Milk, spaghetti, French fries
4. Mineral water, green tea, spaghetti
5. Mineral water, French fries, eggs, soup

위와 같이 정렬되었다면 정렬된 순서대로 tree 를 구축한다.

최종적인 트리는 다음과 같다.



다음으로 만들어놓은 Table 과 생성한 Tree 끼리 같은 item 을 가진 것끼리 Linked List 로 연결시켜주면 FP-Growth 알고리즘의 동작의 원리를 알 수 있다.

<B+ Tree>

B+ Tree는 m 개의 자식과 $m-1$ 개의 데이터를 갖는 다원 트리이다. 기존 다른 Tree의 취약점을 보완하기 위해 고안된 Tree이다.

B+Tree의 가장 큰 특징은 다른 Tree는 보통 데이터를 Root노드부터 Leaf 노드까지 각자 데이터를 가지고 있지만, B+ Tree는 노드의 종류가 DataNode와 IndexNode 두 가지로 나뉜다.

따라서, B+ Tree는 다른 Tree에는 없는 규칙이 존재한다.

첫 번째, 모든 DataNode는 동일한 레벨에 존재하며 Leaf노드에 존재하고 DataNode는 순서대로 서로를 가르키며 연결된다.

두 번째, Data는 Data노드만 가지며 indexNode는 다른 노드를 가리키는 key 값만 가진다.

즉, IndexNode는 탐색하기 위해 사용된다.

세 번째, 만약 노드가 가질 수 있는 정보의 양을 초과하면 해당 노드를 분열해 부모 노드에 값을 전달한다.

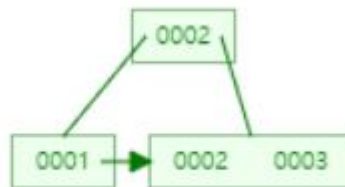
모든 요소들은 Map으로 관리된다.

Index Node는 frequency를 key 값으로 갖고 해당 key 값보다 큰 key값을 가진 노드를 갖는다.

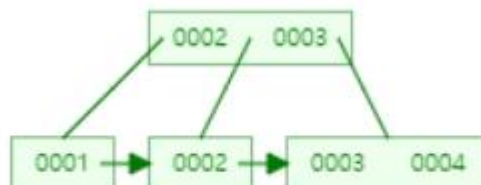
DataNode는 frequency를 key 값으로 갖고 value는 해당 key에 해당하는 frequent pattern을 가진다.

다음은 B+트리가 동작하는 원리이다.

bpointer는 3이라고 가정한다.



위와 같이 B+트리가 만들어져 있을 때 4값을 넣고 싶다면 2와 3을 데이터로 갖는 데이터node가 해당 4값을 가지게 되고 이때 bpointer-1값보다 크므로 해당 데이터 node는 분열한다.



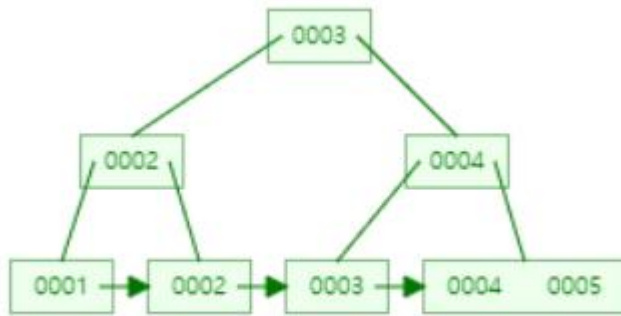
위와 같이 부모 index node에 2,3,4의 중간 값인 3을 전달하고 가장 왼쪽 값인 2는 3,4 데이터

node의 prev로 이동하며 서로 가르킨다. 데이터 노드는 분열할 때마다 prev노드가 존재하지 않으면 새로 생성된 데이터 노드가 원래 있던 노드를 가리키게 하여야 한다.

이는 가장 작은 값을 넣을 때도 같다.

다음은 indexnode가 분열할 때의 사진이다.

위 사진에서 5값을 추가한다고 하면,



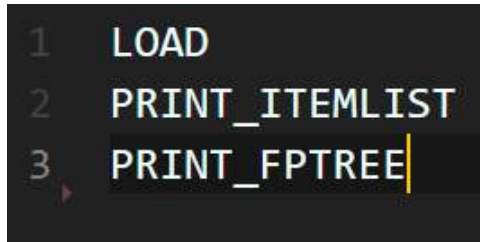
위와 같이 index Node가 분열하게 되는데 3,4,5를 데이터로 갖는 DataNode가 분열하고 그 중간 값인 4가 올라가 indexNode에는 2,3,4의 데이터가 존재한다. 마찬가지로 중간 값인 3이 부모 노드로 올라가게 되고 자식을 2와 3을 갖는다. 이때 주의해야 할 점은 4를 데이터로 갖는 index노드의 LeftMostChild를 3이 가리키고 있던 자식으로 해주어야 한다.

Result screen

검증을 위해 사용할 market.txt와 result.txt는 깃 허브에 올라온 testcase1과 result1을 참고하여 검증하였다.

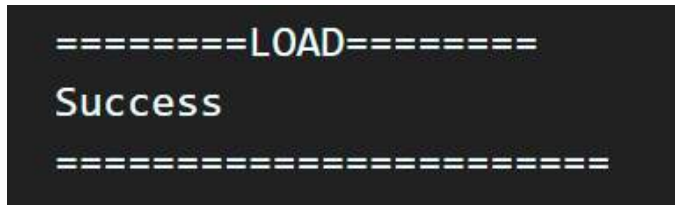
다음은 검증을 위해 사용한 Command.txt와 log.txt의 사진이다.

아래는 Command.txt의 사진이고 LOAD, PRINT_ITEMLIST, PRINT FPTREE 명령어가 있는 것을 확인할 수 있다.



```
1  LOAD
2  PRINT_ITEMLIST
3  PRINT_FPTREE
```

아래는 log.txt의 사진이다. 성공적으로 위 명령어들이 실행된 것을 확인할 수 있다.



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


=====PRINT_ITEMLIST=====

Item	Frequency
------	-----------

soup	12
------	----

spaghetti	9
-----------	---

green tea	9
-----------	---

mineral water	7
---------------	---

milk	5
------	---

french fries	5
--------------	---

eggs	5
------	---

chocolate	5
-----------	---

ground beef	4
-------------	---

burgers	4
---------	---

white wine	3
------------	---

protein bar	3
-------------	---

honey	3
-------	---

energy bar	3
------------	---

chicken	3
---------	---

body spray	3
------------	---

avocado	3
---------	---

whole wheat rice	2
------------------	---

turkey	2
--------	---

shrimp	2
--------	---

pet food	1
----------	---

pepper	1
--------	---

parmesan cheese	1
-----------------	---

meatballs	1
-----------	---

ham	1
-----	---

gums	1
------	---

fresh bread	1
-------------	---

extra dark chocolate	1
----------------------	---

energy drink	1
--------------	---

cottage cheese	1
----------------	---

cookies	1
---------	---

carrots	1
---------	---

bug spray	1
-----------	---

=====

pancakes	2
----------	---

hot dogs	2
----------	---

grated cheese	2
---------------	---

frozen vegetables	2
-------------------	---

frozen smoothie	2
-----------------	---

fresh tuna	2
------------	---

escalope	2
----------	---

brownies	2
----------	---

black tea	2
-----------	---

almonds	2
---------	---

whole wheat pasta	1
-------------------	---

toothpaste	1
------------	---

tomatoes	1
----------	---

soda	1
------	---

shampoo	1
---------	---

shallot	1
---------	---

red wine	1
----------	---

pet food	1
----------	---

pepper	1
--------	---

parmesan cheese	1
-----------------	---

meatballs	1
-----------	---

ham	1
-----	---

gums	1
------	---

```

2 {mineral water.7}
3 (mineral water.1) (green tea.1)
4 (mineral water.3) (spaghetti.5)
5 (mineral water.1) (green tea.2) (spaghetti.5)
6 (mineral water.1) (soup.12)
7 (mineral water.1) (spaghetti.4) (soup.12)
8 {green tea.9}
9 (green tea.1)
0 (green tea.2) (spaghetti.5)
1 (green tea.4) (soup.12)
2 (green tea.2) (spaghetti.4) (soup.12)
3 {spaghetti.9}
4 (spaghetti.5)
5 (spaghetti.4) (soup.12)
6 {soup.12}
7 (soup.12)
8 =====
9

```

위의 사진은 PRINT_FPTree의 사진이며 위 사진에 나온 item을 제외하고 가장 높은 support count를 갖는 item들만 선정하였다.

위 세가지 모두 결과값이 잘 나오는 것을 확인할 수 있다.

다음은 위 세개의 명령어가 에러 코드를 출력할 때의 모습이다.

```

=====LOAD=====
Success
=====

=====LOAD=====
ERROR 100
=====

```

위와 같이 자료구조가 만들어졌는데 또 LOAD 명령어가 들어오면 에러 코드를 출력하게 하였다. 또한 market.txt가 비어 있을 때에도 에러 코드를 출력하게 하였다.

```
4
5  =====PRINT_ITEMLIST=====
6  ERROR 300
7  =====
8
9  =====PRINT_FPTREE=====
10 ERROR 400
11 =====
12
13
```

위 사진은 PRINT_ITEMLIST와 PRINT_FPTREE가 에러 코드를 출력할 때의 사진이다.
각각 header Table과 FP트리가 비어 있을 때 에러 코드를 출력하므로 market.txt를 비워주는 것으
로 에러 코드를 출력하였다.

다음은 BTLOAD의 결과 사진이다.

```
=====BTLOAD=====
Success
=====
```

Result.txt 에 정보가 저장되어 있을 때에는 위와 같이 success 결과를 띄우고

```
1  =====BTLOAD=====
2  Success
3  =====
4
5  =====BTLOAD=====
6  ERROR 200
7  =====
8
```

자료구조가 만들어진 상태에서 한번 BTLOAD 를 실행하면 에러 코드를 출력한다.

또한 result.txt 가 비어 있을 때에는 아래와 같이 에러 코드를 출력한다.

```
=====BTLOAD=====
ERROR 200
=====
```

다음은 PRINT_BPTREE 의 출력 결과이다.

```
=====PRINT_BPTREE=====
FrequentPattern      Frequency
{almonds, eggs} 2
{burgers, eggs} 2
{chicken, eggs} 2
{eggs, french fries} 2
{eggs, mineral water} 2
{eggs, turkey} 2
{almonds, burgers, eggs} 2
{almonds, eggs, soup} 2
{burgers, eggs, soup} 2
{chicken, eggs, mineral water} 2
{eggs, french fries, soup} 2
{almonds, burgers, eggs, soup} 2
{chocolate, eggs} 3
{chocolate, eggs, soup} 3
{eggs, soup} 4
=====
```

위 사진은 PRINT_BPTREE eggs 2 를 실행하였을 때의 출력 결과이다. 각 명령어 사이에는 Tab 으로 구분되어 있다.

Eggs 를 가지는 frequent Pattern 중에 빈도수가 2 가 넘는 것들을 출력하는 것을 확인할 수 있다.

```
1 BTLOAD
2 PRINT_BPTREE eggs|
```

```
1 BTLOAD
2 PRINT_BPTREE 2|
```

```
1 BTLOAD
2 PRINT_BPTREE|
```

```
Assignment 2 > 2 command.txt
1 PRINT_BPTREE eggs 2|
```

위와 같이 명령어 형식에 맞지 않거나 BTLOAD 가 선행되지 않을 상태에서 결과를 얻고자 할 때는 아래와 같이 에러 코드를 출력하게 하였다.

```
=====PRINT_BPTREE=====
ERROR 500
=====
```

다음은 PRINT_CONFIDENCE 의 출력 결과이다.

```

=====PRINT_CONFIDENCE=====
FrequentPattern Frequency      Confidence
{energy bar, milk} 2 0.12
{french fries, milk} 2 0.12
{green tea, milk} 2 0.12
{milk, mineral water} 2 0.12
{milk, soup} 2 0.12
{energy bar, milk, mineral water} 2 0.12
{milk, soup, spaghetti} 2 0.12
{milk, spaghetti} 3 0.18
=====

```

위 사진은 PRINT_CONFIDENCE milk 0.1 을 실행하였을 때의 사진이다.

Milk 의 총 개수를 세서 그 값을 0.1 에 곱하면 해당 부분집합의 빈도수가 나오는데 해당 빈도수보다 높은 빈도수를 가진 부분집합을 출력하였다.

```

1 PRINT_CONFIDENCE milk 1

```

```

1 BTLOAD
2 PRINT_CONFIDENCE milk 1

```

위와 같이 자료구조가 생성되기 전에 명령을 실행하거나 출력할 Frequent Pattern 이 없다면 아래와 같이 에러코드를 출력하게 하였다.

```

1 =====PRINT_CONFIDENCE=====
2 ERROR 600
3 =====
4

```


다음은 PRINT_RANGE의 출력 결과이다.

```
=====PRINT_RANGE=====
FrequentPattern Frequency
{milk, spaghetti} 3
=====
```

위 출력 결과는 PRINT_RANGE milk 3 4를 실행하였을 때의 출력 결과이다.

```
1 BTLOAD
2 PRINT_RANGE milk 100 299
```

```
1 PRINT_RANGE milk 100 299
```

위와 같이 자료구조가 생성되기 전에 명령이 실행되거나 해당 범위에 있는 Frequent Pattern 이 없다면 에러 코드를 출력한다.

```
=====PRINT_RANGE=====
ERROR 700
=====
```

SAVE 명령어는 구현하지 못하였다.

EXIT 명령어는 다음과 같이 실행된다.

```
=====EXIT=====
Success
=====
```

고찰

해당 프로젝트는 FP-Growth 알고리즘과 B⁺-Tree를 이용해 상품 추천 프로그램을 구현하는 것이다. 상품 추천 프로그램을 구현하는 것의 핵심은 FP-Growth 알고리즘과 B⁺-Tree를 이용해 구현하는 것 같았다. 1차 프로젝트인 Linked List와 BST는 직전 학기에 많이 배웠던 것이어서 큰 어려움이 없었지만 이번 2차 프로젝트는 2개의 알고리즘 모두 생소한 알고리즘이라 더 어려웠던 거 같다. 실제로 구현함에 있어 해당 데이터를 가져와 어떻게 Header Table을 만들어야 하며, FP-Tree를 어떻게 구축하는 것에 많은 시간을 썼던 거 같다. 특히 FP-Growth 알고리즘을 구현할 때에 있어서 동시에 Header-Table과 Fp-Tree를 구현하고 싶었지만 둘 사이를 연결시키는 것이 많이 어려워서 Table을 구축한 후에 market.txt 파일을 닫고 다시 열어서 Table을 기준으로 FP-Tree를 구현하였다. 덕분에 데이터가 많은 txt파일을 읽어와서 실행시키면 실행 시간이 압도적으로 늘어나는 것을 확인할 수 있었다.

또한 B⁺ Tree에서는 구현을 다 하였다고 생각하였지만 막상 출력을 확인해보니 segmentation fault가 뜨는 것을 확인할 수 있었다. 이는 데이터 노드를 split 할 때에 이전 노드와 Linked List로 연결시켜주지 않아서 생긴 문제점이었고 만들어진 데이터 노드에 접근 할 수 없어 생긴 문제였다. 해당 문제는 데이터 노드끼리의 연결을 확인하여 다시 연결시켜주었더니 정상적으로 출력되는 것을 확인할 수 있었다. 또한 코딩을 할 때에 조사식을 통해 데이터를 접근해 해당 데이터가 어디에 속해 있는지를 확인하면서 코딩을 하는 스타일인데 이번 프로젝트는 다양한 stl을 사용하여 데이터를 관리하다 보니 조사식을 통해 데이터를 찾으려고 해도 제대로 찾아지지 않았다. 이 문제는 코딩하는 중간중간에 변수를 집어넣어서 해당 데이터가 무슨 값을 갖고 있는지 확인해가며 코딩을 진행하여 해결하였다.

마지막으로 FP-Tree와 B⁺ Tree는 모두 구현하였지만 SAVE 명령어를 통해 두 알고리즘 사이에 연관성을 더해주는 작업을 하고 싶었지만 하지 못해서 아쉬웠다.

또한 새로운 알고리즘을 배워 매우 유익한 프로젝트라고 생각한다.