

데이터구조 2차 프로젝트 결과 보고서



학과 : 컴퓨터정보공학부

학번 : 2021202071

이름 : 이민형

Introduction

이번 프로젝트에서는 FP-Growth와 B+Tree를 이용하여 상품 추천 프로그램을 구현하는 것을 목적으로 한다. 프로그램은 `command.txt`에서 명령어를 입력 받으며, 입력 받은 명령어가 `LOAD`라면 장바구니 데이터에서 같이 구매한 상품들을 받아 FP-Growth를 구축한다. FP-Growth는 상품들의 연관성을 Tree 구조로 저장하고 있는 FP-Tree와 상품별 빈도수와 정보, 해당 상품과 연결된 FP-Tree의 상품 노드들을 관리하는 Header Table로 구성된다.

Frequent Pattern들이 저장된 `result.txt`는 `BTLOAD` 명령어를 통해 빈도수를 기준으로 B+Tree에 저장된다. B+Tree는 `IndexNode`와 `DataNode`로 구성된다. `BpTreeIndexNode`와 `BpTreeDataNode`는 `BpTreeNode`를 상속받으며 Polymorphsim을 통해서 구현한다. 이는 객체지향 언어에서는 어떤 클래스의 포인터 변수에 그 클래스의 하위 클래스 객체의 주소도 할당할 수 있도록 허용되기 때문이다. `IndexNode`는 `DataNode`를 찾기 위한 Node이고, `DataNode`는 해당 빈도수를 가지는 Frequent Pattern들이 저장된 Node이다.

다음은 프로그램에서 구현하는 자료구조에 대해서 알아보도록 하겠다.

1) FP-Growth

- 주어진 `market.txt`에 저장된 데이터를 이용하여 구축한다. `market.txt`에서 같이 구매한 물품은 줄 단위로 구분되어 있다.
- 프로그램 구현 시 주어진 `FPNode` 클래스를 통해 FP-Tree를 구현하고 주어진 `HeaderTable` 클래스를 통해서 `HeaderTable`을 구현한다. 상품 정보는 항상 고유하며, 소문자로 표기한다.
- FP-Growth의 threshold값은 고정되어 있지 않으며 멤버변수로 변경할 수 있다. threshold 값은 2 이상으로 설정한다고 가정한다.

2) Header Table

- Header Table은 `indexTable`과 `dataTable`로 구성된다. Header Table에는 threshold보다 작은 상품들도 저장한다.
- `indexTable`은 빈도수를 기준으로 정렬된 상품들을 저장하는 변수이며, `list`를 통해서 구현한다. `list`에는 인자로 `pair`를 통해 빈도수와 상품명을 저장한다.

- dataTable은 상품명과 상품에 연결되는 포인터를 저장하는 변수이다. stl map을 통해서 구현하며 key에는 상품명, value에는 상품에 연결되는 FP-Tree의 node 포인터를 저장한다.

3) FP-Tree

- FP-Tree 클래스는 따로 생성하지 않고 FPNODE를 이용하여 구축한다. root에서 자식 노드를 제외한 변수는 NULL값을 가진다.
- 자식 노드들은 map 행태로 저장하며, 각 Childrennode들은 부모 노드를 가리킨다. 자식 노드를 관리하는 map은 key값으로 상품명, value값으로 해당 상품의 빈도수 정보 및 연결된 Node 정보들을 저장한다.
- FP-Tree에 저장된 노드들은 Header Table에서 같은 상품 노드들끼리 연결되어야 한다.

4) B+Tree

- result.txt에 저장된 데이터를 이용하여 구축한다. B+Tree는 빈도수를 기준으로 정렬된다.
- DataNode는 단말노드로 해당 빈도수를 Key값으로 가지고, 해당 빈도수에 속하는 Frequent Pattern이 저장된 FrequentPatternNode를 Value값으로 가지는 mapData를 가지고 있다.
- IndexNode는 datanode를 찾기 위한 node로 빈도수를 key 값으로 가지고, 그의 자식 노드인 BpTreenode를 Value값으로 가진다. 또한 가장 왼쪽 자식을 가리키는 포인터인 pMostLeftChild를 따로 가진다.
- B+Tree의 Order인 bpOrder은 고정되어 있지 않으며 멤버 변수로 변경할 수 있다.

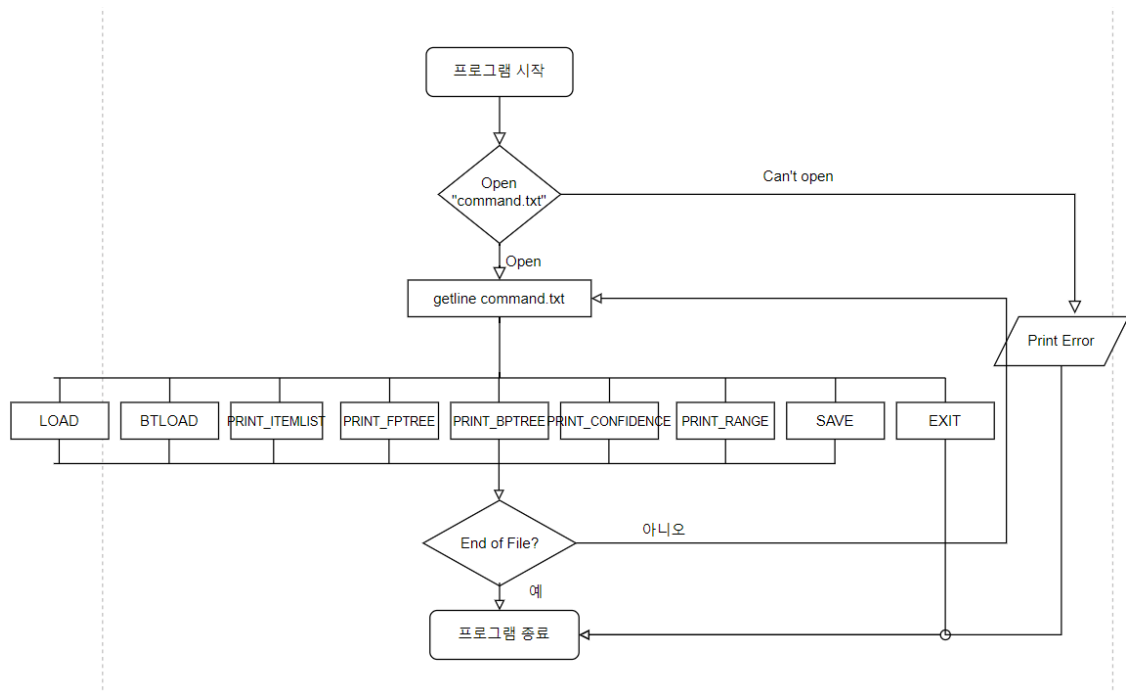
5) FrequentPatternNode

- FrequentPatternNode는 Frequent Pattern 정보를 아래와 같이 multimap 컨테이너 형태로 가진다. Key값에는 Datanode에서 가지는 Frequency값을 가지며, Value값에는 set컨테이너로 Frequent Pattern 정보를 저장한다.
- Frequent Pattern 중 공집합이거나 원소가 하나인 집합은 저장하지 않는다.

Flow Chart

Run

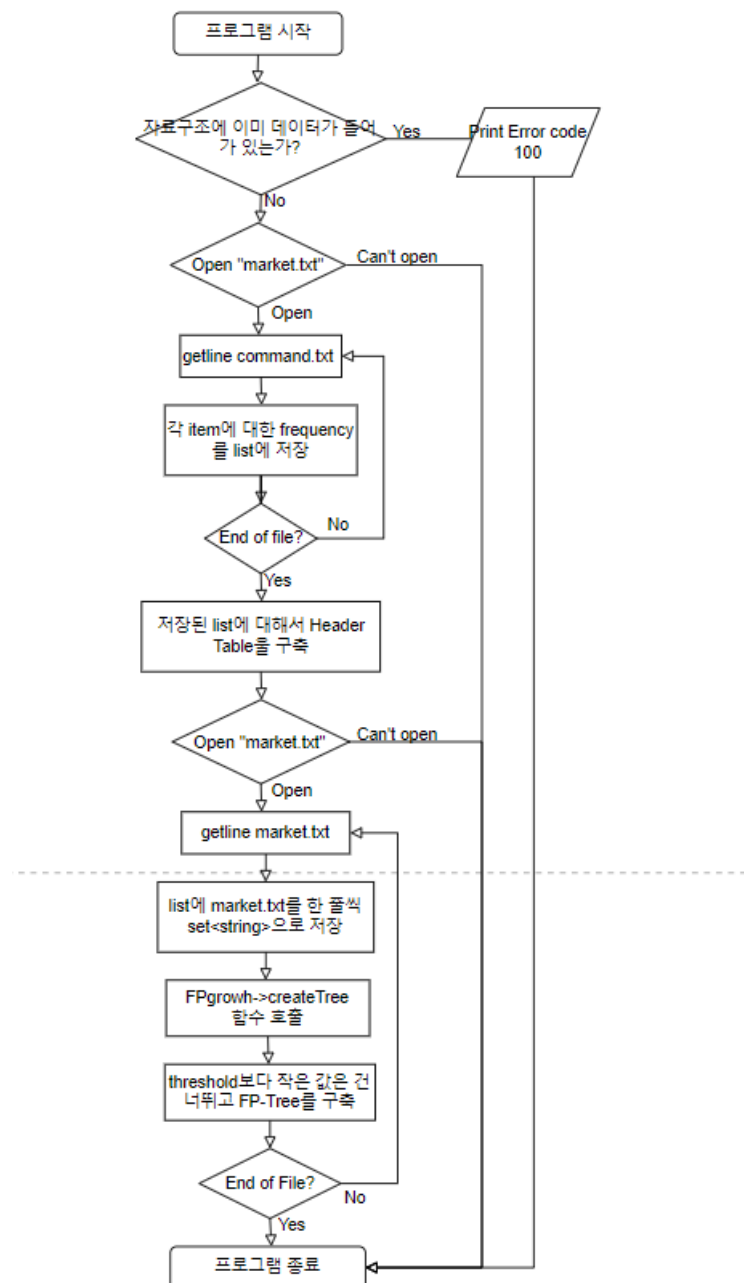
우선 본 프로젝트에서 구현하는 전체적인 클래스를 관리하는 manager 클래스에서 command.txt파일에서 명령어를 읽어 실행하는 run함수의 Flow Chart는 다음과 같다.



<Run 함수의 Flow Chart>

LOAD

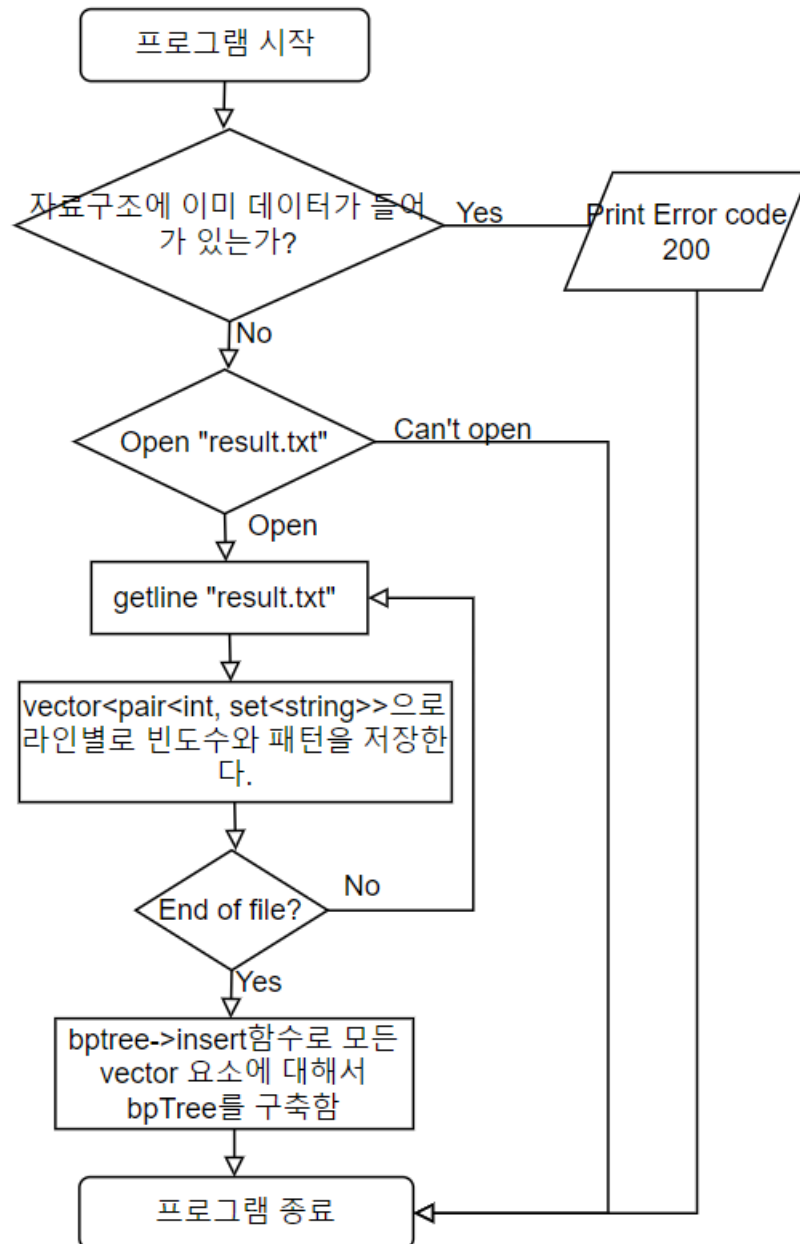
다음은 LOAD함수의 Flow chart이다. LOAD함수는 market.txt에서 파일을 읽어와 FP-Growth를 구축하는 명령어이다. 만약 이전에 LOAD가 되었다면 에러 메시지를 출력하고 프로그램을 종료한다.



<LOAD 명령어의 Flow Chart>

BTLOAD

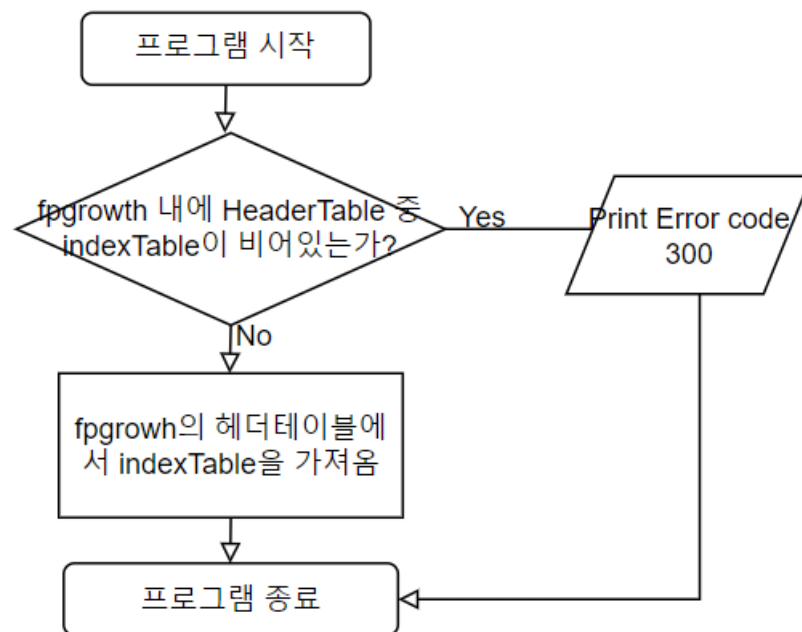
다음은 BTLOAD 명령어이다. BTLOAD는 result.txt에 대하여 BpTree를 구축하는 명령어이다. BpTree에 이미 데이터가 들어있는 상태에서 명령어를 실행하면 에러 메시지를 출력하고 프로그램을 종료한다.



<BTLOAD 명령어의 Flow Chart>

PRINT_ITEMLIST

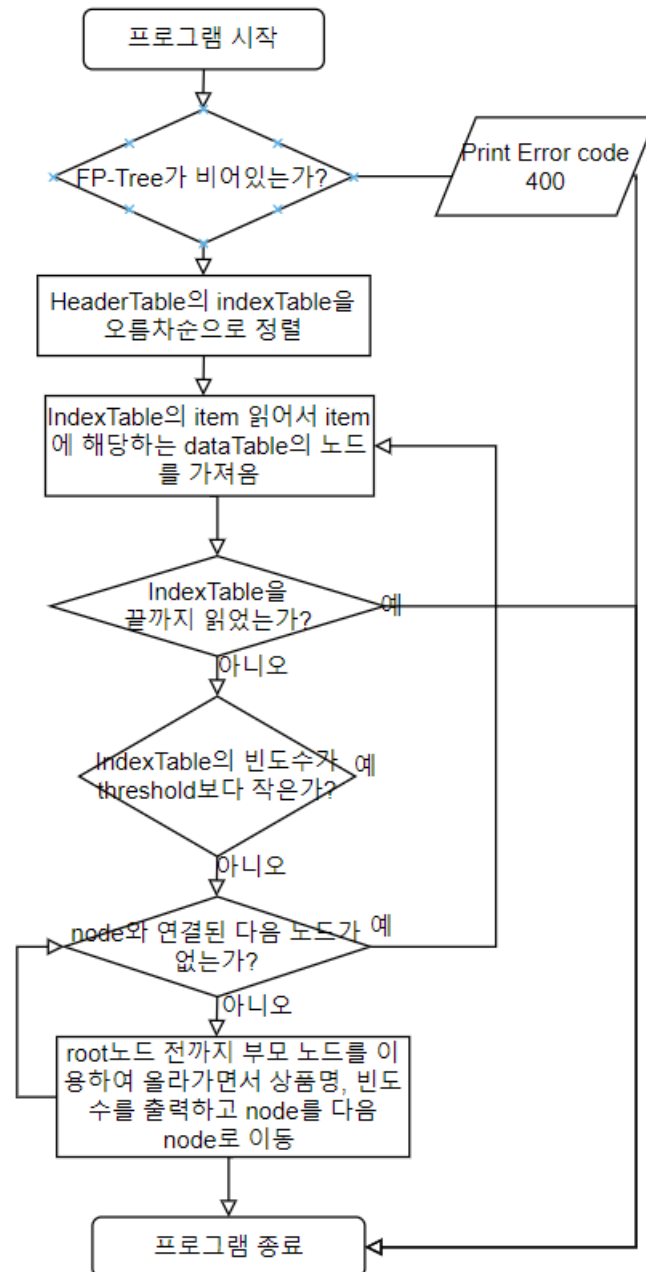
다음은 PRINT_ITEMLIST 명령어의 Flow Chart이다. FP-Growth의 Header Table에 저장된 상품들을 빈도수를 기준으로 내림차순으로 출력하는 명령어이다. threshold보다 작은 값도 출력한다. HeaderTable이 비어 있는 경우 에러코드를 출력한다.



<PRINT_ITEMLIST의 Flow Chart>

PRINT_FPTREE

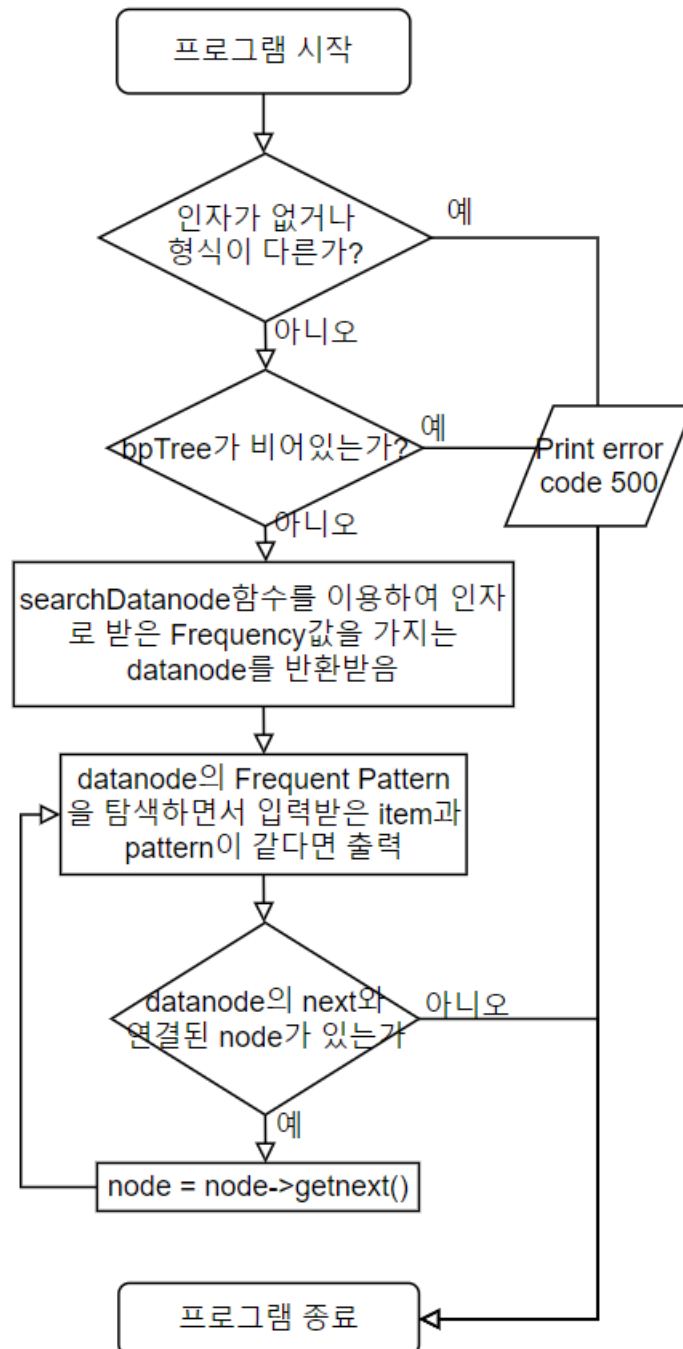
다음은 PRINT_FPTREE 명령어의 Flow Chart이다. 헤더테이블을 빈도수를 기준으로 오름차순 정렬하고 threshold 이상의 상품들을 출력한다. FP-Tree가 비어 있는 경우 에러코드를 출력한다.



<PRINT_FPTREE의 Flow Chart>

PRINT_BPTREE

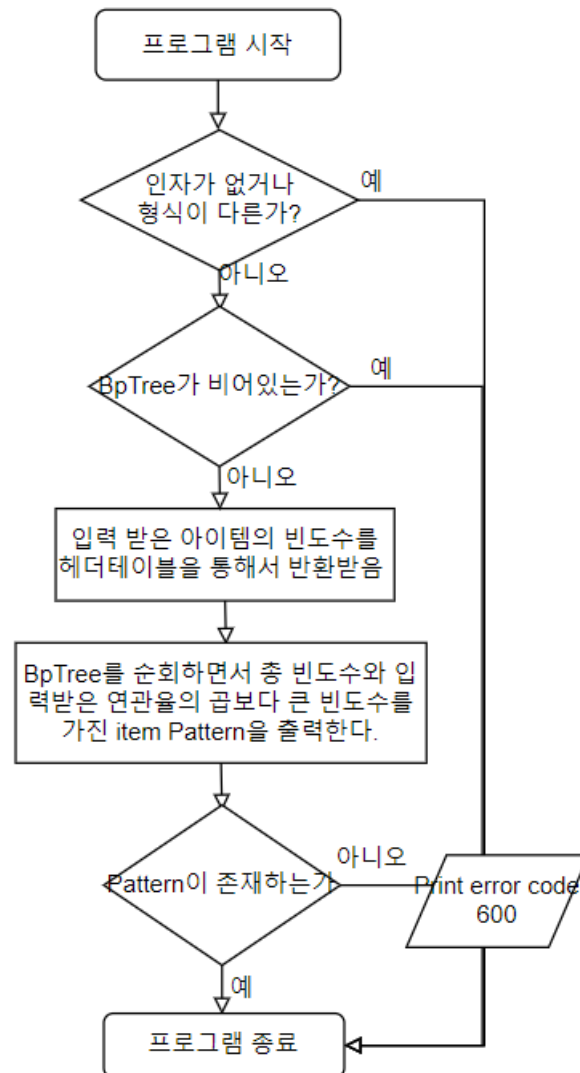
다음은 PRINT_BPTREE 명령어이다. 최소 빈도수를 기준으로 bpTree에서 탐색한다. bpTree에서 최소 빈도수를 만족하며, 입력된 상품을 포함하는 Frequent Pattern을 출력하며 이동한다. 출력할 Frequent Pattern이 없거나 bpTree가 비어 있는 경우, 그리고 인자가 부족하거나 형식이 다른면 에러 코드를 출력한다.



<PRINT_BPTREE의 Flow Chart>

PRINT_CONFIDENCE

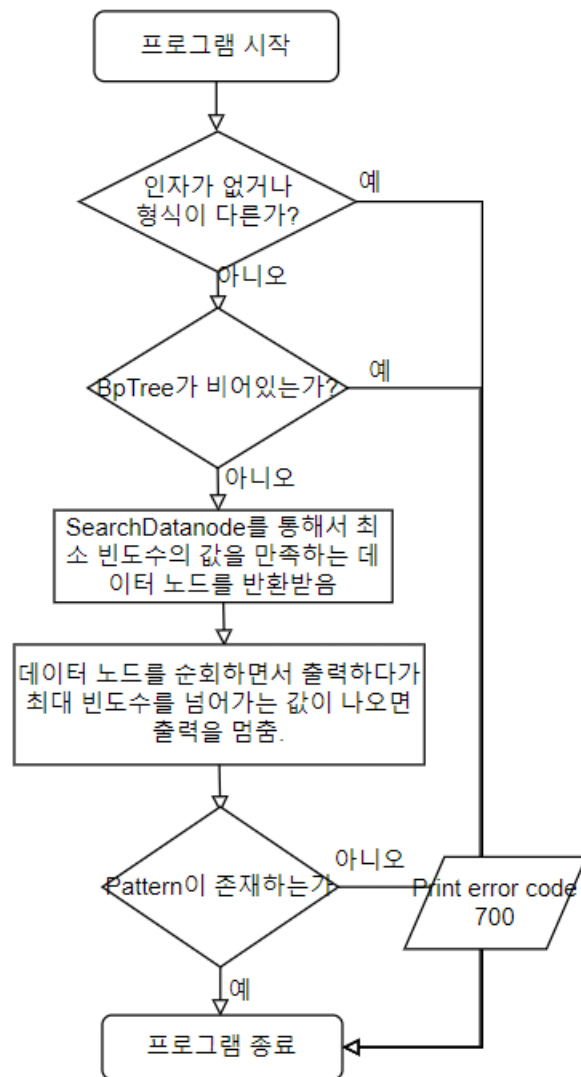
다음은 PRINT_CONFIDENCE함수의 Flow Chart이다. BpTree에 저장된 Frequent Pattern 중 입력된 상품과 연관율 이상의 값을 가지는 Frequent Pattern을 출력하는 명령어이다. 인자가 모두 입력되지 않거나 형식이 다를 때 에러코드를 출력한다. 또한, 출력할 Frequent이 없거나 BpTree가 비어있는 경우 에러코드를 출력한다.



<PRINT_CONFIDENCE 함수의 Flow Chart>

PRINT_RANGE

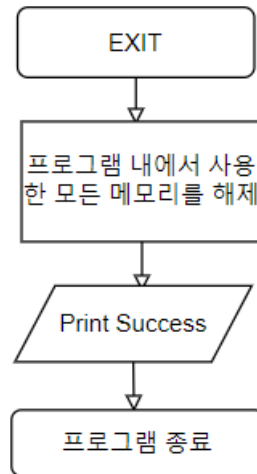
다음은 PRINT_RANGE 함수의 Flow Chart이다. 첫 번째 인자로 상품명을 입력 받고, 두 번째 인자로 최소 빈도수, 세 번째 인자로 최대 빈도수를 입력 받아 최소 빈도수부터, 최대 빈도수까지 탐색을 하여 해당 아이템이 존재한다면 패턴을 출력한다. 인자가 모두 입력되지 않거나 형식이 다르다면 에러코드를 출력한다. 또한 출력할 Pattern이 없거나 BpTree가 비어 있는 경우 에러코드를 출력한다.



<PRINT_RANGE의 Flow Chart>

EXIT

마지막으로 EXIT 명령어이다. EXIT 명령어는 프로그램 내에서 사용한 모든 메모리를 해제하고, run함수를 종료한다.



<EXIT의 Flow Chart>

Algorithm

본 프로젝트에서는 FP-growth와 BpTree의 알고리즘을 사용한다.

FP-Growth

먼저 FP-Growth는 어떤 사건이 얼마나 자주 함께 발생하는지, 서로 얼마나 연관되어 있는 지를 표시하는 Association Rule을 찾는 데에 사용한 방법 중 하나이다. 우선 Association Rule에 대해서 설명하도록 하겠다.

Association Rule에는 사건이 얼마나 함께 자주 발생하는지를 측정할 수 있는 3가지 척도가 있다. 우선 첫 번째로 Support이다. Support는 전체 경우의 수에서 두 아이템이 같이 나오는 비율을 의미한다. 다음은 Confidence이다. Confidence는 X가 나온 경우 중 X와 Y가 함께 나올 비율을 의미한다. 즉, 구매의 예로 들자면 사과를 사고, 감자도 같이 사는 사람의 비율을 말한다. 다음은 설계 시간에는 다루지 않았지만 lift이다. Lift는 X와 Y가 같이 나오는 비율을 X가 나올 비율과 Y가 나올 비율의 곱으로 나눈 값이다.

우선 FP-growth를 구축할 때 해야할 일은 Header Table을 구축하는 것이다. Header Table은 IndexTable과 dataTable로 구성된다. IndexTable은 각 상품을 빈도수 별로 저장한 것이고, dataTable은 각 상품과 상품에 연결되는 FP-Tree의 Node 포인터를 저장한다.

다음은 FP-Tree를 구축하는 과정이다. CreateFPtree함수를 통해서 FP-Tree를 구축한다. Insert 함수를 통하여 market.txt의 한 줄인 item_array와 FP-Tree의 root노드, HeaderTable의 테이블을 인자로 받아서 구현한다. 우선 item_array가 들어오면 IndexTable을 기준으로 빈도수가 높은 순서대로 정렬을 진행한다.

예를 들어 IndexTable이 다음과 같이 존재한다고 하자.

Item	Frequency
Milk	8
Eggs	6
Hot dog	3
Soup	2
Chocolate Chip	1

그런데 item_array가 Chocolate Chip, Hot dog, Eggs, Soup 순서로 들어왔다고 하면, FP-Tree에는 threshold보다 작은 값은 저장하지 않기 때문에 Chocolate Chip은 무시하고 빈도수가 큰 순서대로 정렬한다. 따라서 정렬된 item_array는 다음과 같다. Eggs, Hot dog, Soup 순서로 FP-Tree를 구축한다. 우선 root 노드는 자식 노드를 제외한 변수들은 NULL값을 가진다. FPNODE 변수 p를 root로 두고 정렬된 item_array의 사이즈만큼 반복문을 수행한다. 반복문을 수행하여 루트 노드부터 루트 노드의 Children을 탐색하여 만약 해당 item을 가지는 Children node가 존재한다면 p를 루트 노드의 Children node로 이동하여 Children node의 빈도수를 1만큼 증가시키고 다시 반복문을 수행한다. 반면 루트 노드의 Children을 탐색하였는데 해당 item을 가지는 Children node가 존재하지 않는다면 FPNODE를 새로 만들고, 새로 만든 node의 빈도수를 1로 두고, 새로 만든 node의 부모를 p로 설정한다. p는 자식들을 Map으로 관리하기 때문에 Map에 item과 새로 만든 FPNODE를 Insert한다. 그런 뒤 p를 새로 삽입한 node로 이동시킨다. 따라서 현재 p node는 이전에 삽입했거나, 빈도수를 증가시켜주었던 node를 가리키게 된다. 이를 item_array만큼 반복하게 되면 market.txt의 한 줄에 대한 FP-Tree를 구현하게 된 것이다. 이를 market.txt의 모든 라인에 대해 반복하면 FP-Tree를 구축할 수 있다.

다음은 dataTable의 FPNODE와 FP-Tree의 노드들간의 연결을 하는 과정이다. 사실 이는 FP-Tree를 구축하면서 동시에 이루어지게 되는데, connectNode 함수의 인자는 HeaderTable, item 그리고 연결을 진행해줄 node로 이루어지게 된다. 노드 간의 연결은 새로운 node를 만들어줄때에만

연결을 수행해주면 되기 때문에 새로 노드를 만들고, map에 노드를 넣고 나서 connectNode 함수를 호출한다. connectNode함수의 node인자에는 새로 만든 노드가 인자로 주어지게 된다. connectNode 함수는 인자로 받은 item을 이용하여 dataTable에서의 FPNODE를 받아온다. 받아온 node의 next가 NULL이라면 table과 인자로 받은 node를 연결시키고 함수를 종료하면 된다. 반면 dataTable에서 받아온 node의 next가 NULL이 아니라면 해당 노드의 next가 NULL일때까지 노드를 다음 노드로 이동한다. node의 next가 NULL일때까지 이동하였다면 node와 인자로 받아온 node를 연결한 뒤, 함수를 종료한다.

B+Tree

다음은 본 프로젝트에서 구현하는 B+Tree 알고리즘에 대해서 알아보도록 하겠다. B+Tree는 동작 방식이 B-Tree와 굉장히 유사하지만 다른 점이라고 하면 leaf node가 연결리스트의 형태를 띄어 선형 검색이 가능한 점에서 차이점이라고 할 수 있다. 이러한 특징 때문에 B-Tree와 비교하여 상대적으로 작은 시간 복잡도로 검색을 수행할 수 있다. 따라서 B+Tree는 B-Tree를 개선한 형태라고도 할 수 있다.

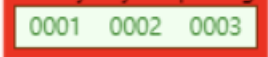
B-Tree와 B+Tree의 차이점을 살펴보면 다음과 같다.

- BpTree에는 인덱스와 데이터라는 두 가지 유형의 노드가 있습니다.
- BpTree의 인덱스 노드는 B-tree의 Internal노드에 해당하는 반면 데이터 노드는 External노드에 해당합니다.
- 인덱스 노드는 키(요소가 아닌)와 포인터를 저장하고 데이터 노드는 요소(키와 함께)를 저장하지만 포인터는 저장하지 않습니다.
- 데이터 노드들은 서로 연결되어 이중 연결 리스트를 형성한다. 리스트는 정렬된 상태를 유지한다.
- 모든 데이터 노드는 동일한 Level에 있으며 leaf이다. 데이터 노드에는 요소만 포함된다. 인덱스 노드는 bpOrder 3의 B-Tree와 동작이 같다. 각 인덱스 노드에는 키만 존재한다.

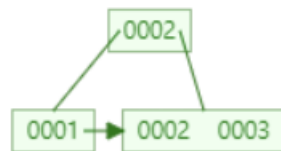
본 프로젝트에서 BpTree를 구축하기 위해서 총 3가지 함수를 구현한다. 첫 번째로 IndexNode를 탐색하여 Insert해주어야 하는 Datanode를 찾아서 Datanode에 삽입하는 Insert 함수이다. Datanode에 Insert를 해준 뒤, Datanode가 최대 노드의 개수를 초과하지 않았는지 확인을 하여, 초과하였다면 Datanode Split을 진행하여 준다. Datanode Split을 진행하는 함수가 splitDataNode

함수이다. splitDataNode는 크게 두가지 경우로 이루어진다. 입력 받은 인자가 root node인 경우와 아닌 경우이다. 우선, 인자로 입력 받은 Datanode가 root 노드인 경우이다.

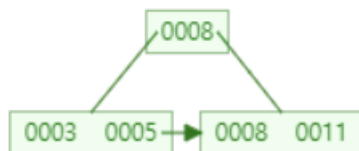
Node now contains too many keys. Splitting ...



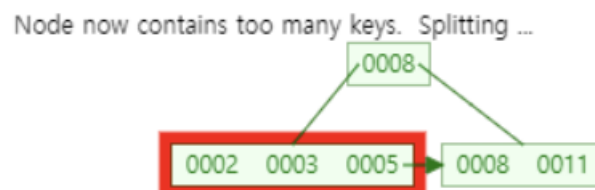
위의 사진은 BpOrder가 3일 때 원소의 최대 개수는 2개이므로 Root node가 Split되는 경우이다. Split이 되는 index는 Split하려는 dataNode의 원소의 개수에서 2를 나누고, 소수점 자리는 버리는 방식을 사용하였다. 또한 Split을 구현한 뒤, Split된 노드들의 부모를 새로운 루트로 정의하고 양방향연결을 하여준다. 위의 경우를 Split을 진행하면 다음 사진과 같다.



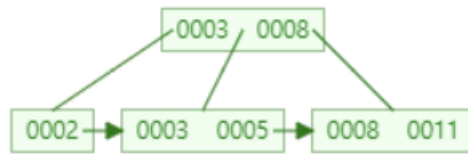
다음은 입력받은 Datanode가 root node가 아닌 경우에서의 DataSplit이다.



위의 사진에서 Key값이 2인 node를 Insert하게 되면 다음 사진과 같다.



Key값을 2,3,5를 가지는 node가 Split되어야 한다. 3의 Key가 Datanode의 Parent로 Insert되고, 3은 3, 5를 자식으로 가진다. 루트 노드를 Split할 때와 마찬가지로 Split된 노드들의 부모와 양방향 연결을 수행한 후의 사진은 다음과 같다.



SplitDataNode가 수행되고 나면, Split된 Node의 Parent 노드가 최대 원소의 개수, 즉 bpOrder-1의 값을 넘지 않았는 지를 확인하여 넘었다면 splitIndexNode함수를 수행한다.

splitIndexNode함수도 splitDataNode함수와 마찬가지로 Split하려는 IndexNode가 Rootnode인 경우와 아닌 경우로 나누어서 구현하였다. 우선 Split하려는 IndexNode가 root 노드인 경우는 다음 사진과 같다. 이전의 DataSplit이 이루어진 상황에서 키 값이 2인 Node가 새로 들어온다면 DataSplit이 먼저 일어나고, 그 node의 Parent인 IndexNode로 값이 들어가게 된다. Parent가 root인 상황에서 최대 원소의 개수를 초과하기 때문에 IndexNode Split이 이루어지게 된다. IndexNode Split이 이루어지기 전까지의 상황은 다음과 같다.

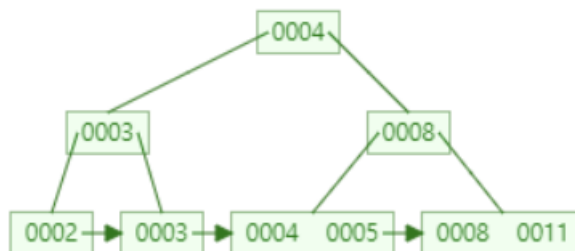
Node now contains too many keys. Splitting ...



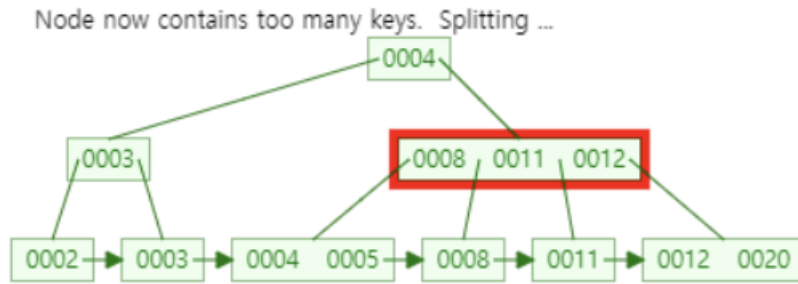
Node now contains too many keys. Splitting ...



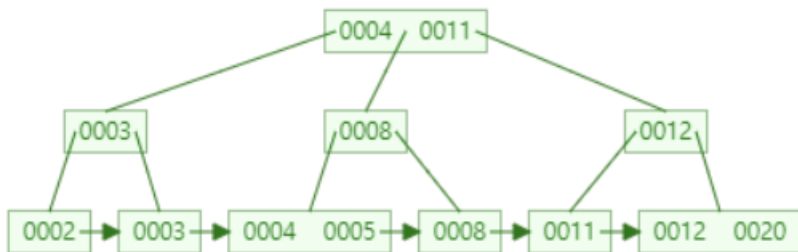
다음은 Root 노드인 IndexNode가 Split되는 상황이다. IndexNode는 B-Tree의 Split 방식과 같기 때문에 3,4,8의 IndexNode를 스플릿 할 때 3이 왼쪽으로 잘리고, 8이 오른쪽으로 잘린다. 4,8이 오른쪽으로 잘리는 DataNode Split과 차이가 있다. 새로운 루트를 선언하고 Key가 4인 자식은 오른쪽으로 스플릿 된, 즉 Key가 8인 node의 MostleftChild가 된다. Split이 이루어진 상황은 다음 사진과 같다.



다음은 Split하려는 IndexNode가 Root 노드가 아닌 경우에 대하여 Split하는 경우는 다음 사진과 같다.



Key값이 8,11,12인 IndexNode의 Split을 수행한다. 우선 parent node로 갈 Key는 11이고, 왼쪽으로 Split될 node의 Key값은 8이고, 오른쪽으로 Split될 node의 Key값은 12이다. Key값이 11인 node의 자식은 오른쪽으로 Split된 node의 MostLeftChild가 된다.



Split이 모두 수행된 사진은 위와 같다. Split할 IndexNode가 root 노드가 아닌 경우에서 Split을 마친 뒤, Indexnode의 Parentnode가 최대 원소의 개수를 넘어가는지를 확인하여 다시 재귀적으로 함수를 수행하여 모든 Split을 마친다.

Result Screen

(testcase2, result2)와 (testcase3, result3)는 데이터의 수가 많아서 testcase1과 result1으로 결과화면을 출력하였다.

우선 LOAD 명령어의 결과화면이다. LOAD명령어는 FP-Growth를 구축하는 명령어로 만약 이미 자료구조에 데이터가 들어가 있다면 에러코드를 출력한다.

```

peter@ubuntu:~/Desktop/DS_project2_draft/DS_Projec
=====LOAD=====
Success
=====

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

다음은 BTLOAD 명령어의 결과화면이다. BTLOAD 명령어는 B+Tree를 구축하는 명령어로, 만약 이미 자료구조에 데이터가 들어가 있으면 에러코드를 출력한다.

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

다음은 PRINT_ITEMLIST 명령어의 결과화면이다. PRINT_ITEMLIST는 FP-growth의 헤더테이블 중 indexTable을 출력하는 명령어로 오름차순으로 정렬한 뒤 출력한다. 만약 LOAD이전에, 즉 HeaderTable이 만들어지기 이전에 명령어가 실행되면 에러코드를 출력한다.

```
=====PRINT_ITEMLIST===
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
salmon 2
pasta 2
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
fresh bread 1
extra dark chocolate 1
energy drink 1
cottage cheese 1
cookies 1
carrots 1
bug spray 1
=====
```

```
=====PRINT_ITEMLIST===
ERROR 300
=====
```

다음은 PRINT_FPTREE 명령어이다. FP-Tree의 Leaf node부터 차례로 부모 노드로 이동하여 root 노드 이전까지 출력한다. FP-Tree가 비어 있는 경우에는, 즉 BTLOAD보다 이전에 실행하면 에러코드를 출력한다.

```

=====PRINT_FPTREE=====
{StandardItem, Frequency} (path_Item, Frequency)
{almonds, 2}
(almonds, 1)(burgers, 1)(eggs, 1)(french fries, 2)(green tea, 4)(soup, 12)
(almonds, 1)(hot dogs, 1)(turkey, 1)(burgers, 1)(ground beef, 1)(chocolate, 2)(eggs, 2)(soup, 12)
{black tea, 2}
(black tea, 1)(fresh tuna, 1)(salmon, 1)(turkey, 1)(chicken, 1)(eggs, 1)(mineral water, 3)(spaghetti, 5)
(black tea, 1)(escalope, 1)(frozen smoothie, 1)(salmon, 1)(energy bar, 1)(ground beef, 1)(milk, 1)(mineral water, 3)(spaghetti, 5)
{brownies, 2}
(brownies, 1)(hot dogs, 1)(pancakes, 1)(avocado, 1)(body spray, 1)(french fries, 2)(green tea, 4)(soup, 12)
(brownies, 1)(white wine, 1)(chocolate, 1)(green tea, 2)(spaghetti, 5)
{escalope, 2}
(escalope, 1)(frozen smoothie, 1)(salmon, 1)(energy bar, 1)(ground beef, 1)(milk, 1)(mineral water, 3)(spaghetti, 5)
(escalope, 1)(fresh tuna, 1)(frozen smoothie, 1)(frozen vegetables, 1)(whole wheat rice, 1)(honey, 1)(mineral water, 1)(spaghetti, 4)(soup, 12)
{fresh tuna, 2}
(fresh tuna, 1)(salmon, 1)(turkey, 1)(chicken, 1)(eggs, 1)(mineral water, 3)(spaghetti, 5)
(fresh tuna, 1)(frozen smoothie, 1)(frozen vegetables, 1)(whole wheat rice, 1)(honey, 1)(mineral water, 1)(spaghetti, 4)(soup, 12)
{frozen smoothie, 2}
(frozen smoothie, 1)(salmon, 1)(energy bar, 1)(ground beef, 1)(milk, 1)(mineral water, 3)(spaghetti, 5)
(frozen smoothie, 1)(frozen vegetables, 1)(whole wheat rice, 1)(honey, 1)(mineral water, 1)(spaghetti, 4)(soup, 12)
{frozen vegetables, 2}
(frozen vegetables, 1)(whole wheat rice, 1)(honey, 1)(mineral water, 1)(spaghetti, 4)(soup, 12)
(frozen vegetables, 1)(ground beef, 1)(chocolate, 1)(green tea, 2)(spaghetti, 4)(soup, 12)
{grated cheese, 2}
(grated cheese, 1)(pasta, 1)(shrimp, 1)(avocado, 1)(honey, 1)(white wine, 1)(burgers, 1)
(grated cheese, 1)(white wine, 1)(ground beef, 1)(mineral water, 3)(spaghetti, 5)
{hot dogs, 2}
(hot dogs, 1)(pancakes, 1)(avocado, 1)(body spray, 1)(french fries, 2)(green tea, 4)(soup, 12)
(hot dogs, 1)(turkey, 1)(burgers, 1)(ground beef, 1)(chocolate, 2)(eggs, 2)(soup, 12)
{pancakes, 2}
(pancakes, 1)(body spray, 1)(mineral water, 1)(green tea, 2)(spaghetti, 5)
(pancakes, 1)(avocado, 1)(body spray, 1)(french fries, 2)(green tea, 4)(soup, 12)

```

```

{pasta, 2}
(pasta, 1)(shrimp, 1)(chocolate, 2)(eggs, 2)(soup, 12)
(pasta, 1)(shrimp, 1)(avocado, 1)(honey, 1)(white wine, 1)(burgers, 1)
{salmon, 2}
(salmon, 1)(turkey, 1)(chicken, 1)(eggs, 1)(mineral water, 3)(spaghetti, 5)
(salmon, 1)(energy bar, 1)(ground beef, 1)(milk, 1)(mineral water, 3)(spaghetti, 5)
{shrimp, 2}
(shrimp, 1)(chocolate, 2)(eggs, 2)(soup, 12)
(shrimp, 1)(avocado, 1)(honey, 1)(white wine, 1)(burgers, 1)
{turkey, 2}
(turkey, 1)(chicken, 1)(eggs, 1)(mineral water, 3)(spaghetti, 5)
(turkey, 1)(burgers, 1)(ground beef, 1)(chocolate, 2)(eggs, 2)(soup, 12)
{whole wheat rice, 2}
(whole wheat rice, 1)(energy bar, 1)(milk, 1)(mineral water, 1)(green tea, 1)
(whole wheat rice, 1)(honey, 1)(mineral water, 1)(spaghetti, 4)(soup, 12)
{avocado, 3}
(avocado, 1)(honey, 1)(white wine, 1)(burgers, 1)
(avocado, 1)(milk, 1)(spaghetti, 4)(soup, 12)
(avocado, 1)(body spray, 1)(french fries, 2)(green tea, 4)(soup, 12)
{body spray, 3}
(body spray, 1)(mineral water, 1)(green tea, 2)(spaghetti, 5)
(body spray, 1)(french fries, 2)(green tea, 4)(soup, 12)
(body spray, 1)(chicken, 1)(green tea, 4)(soup, 12)
{chicken, 3}
(chicken, 1)(eggs, 1)(mineral water, 3)(spaghetti, 5)
(chicken, 1)(chocolate, 1)(eggs, 1)(french fries, 1)(mineral water, 1)(soup, 12)
(chicken, 1)(green tea, 4)(soup, 12)
{energy bar, 3}
(energy bar, 1)(milk, 1)(mineral water, 1)(green tea, 1)
(energy bar, 1)(ground beef, 1)(milk, 1)(mineral water, 3)(spaghetti, 5)
(energy bar, 1)(protein bar, 1)(soup, 12)
{honey, 3}
(honey, 1)(protein bar, 1)(french fries, 1)(milk, 1)
(honey, 1)(white wine, 1)(burgers, 1)
(honey, 1)(mineral water, 1)(spaghetti, 4)(soup, 12)
{protein bar, 3}
(protein bar, 1)(french fries, 1)(milk, 1)
(protein bar, 1)(green tea, 4)(soup, 12)
(protein bar, 1)(soup, 12)

```

```

{white wine, 3}
(white wine, 1)(burgers, 1)
(white wine, 1)(chocolate, 1)(green tea, 2)(spaghetti, 5)
(white wine, 1)(ground beef, 1)(mineral water, 3)(spaghetti, 5)
{burgers, 4}
(burgers, 1)
(burgers, 1)(french fries, 1)(milk, 1)(green tea, 2)(spaghetti, 4)(soup, 12)
(burgers, 1)(eggs, 1)(french fries, 2)(green tea, 4)(soup, 12)
(burgers, 1)(ground beef, 1)(chocolate, 2)(eggs, 2)(soup, 12)
{ground beef, 4}
(ground beef, 1)(milk, 1)(mineral water, 3)(spaghetti, 5)
(ground beef, 1)(mineral water, 3)(spaghetti, 5)
(ground beef, 1)(chocolate, 1)(green tea, 2)(spaghetti, 4)(soup, 12)
(ground beef, 1)(chocolate, 2)(eggs, 2)(soup, 12)
{chocolate, 5}
(chocolate, 2)(eggs, 2)(soup, 12)
(chocolate, 1)(eggs, 1)(french fries, 1)(mineral water, 1)(soup, 12)
(chocolate, 1)(green tea, 2)(spaghetti, 5)
(chocolate, 1)(green tea, 2)(spaghetti, 4)(soup, 12)
{eggs, 5}
(eggs, 1)(mineral water, 3)(spaghetti, 5)
(eggs, 2)(soup, 12)
(eggs, 1)(french fries, 1)(mineral water, 1)(soup, 12)
(eggs, 1)(french fries, 2)(green tea, 4)(soup, 12)
{french fries, 5}
(french fries, 1)(milk, 1)
(french fries, 1)(mineral water, 1)(soup, 12)
(french fries, 2)(green tea, 4)(soup, 12)
(french fries, 1)(milk, 1)(green tea, 2)(spaghetti, 4)(soup, 12)
{milk, 5}
(milk, 1)(mineral water, 1)(green tea, 1)
(milk, 1)
(milk, 1)(spaghetti, 4)(soup, 12)
(milk, 1)(mineral water, 3)(spaghetti, 5)
(milk, 1)(green tea, 2)(spaghetti, 4)(soup, 12)

```

```

{mineral water, 7}
(mineral water, 1)(green tea, 1)
(mineral water, 3)(spaghetti, 5)
(mineral water, 1)(green tea, 2)(spaghetti, 5)
(mineral water, 1)(soup, 12)
(mineral water, 1)(spaghetti, 4)(soup, 12)
{green tea, 9}
(green tea, 1)
(green tea, 2)(spaghetti, 5)
(green tea, 4)(soup, 12)
(green tea, 2)(spaghetti, 4)(soup, 12)
{spaghetti, 9}
(spaghetti, 5)
(spaghetti, 4)(soup, 12)
{soup, 12}
(soup, 12)
=====

```

```

=====PRINT_FPTREE=====
ERROR 400
=====

```

다음은 PRINT_BPTREE 명령어이다. 이 명령어는 BPTREE를 탐색하여 해당 빈도수 이상의 값을 만족하며, 입력된 상품을 포함하는 Frequent Pattern을 출력하며 이동한다. 출력할 패턴이 없거나, BpTree가 비어 있거나, 인자의 형식이 다르면 에러코드를 출력한다. 아래의 결과화면은 soup 2로 결과화면을 출력한 것이다.

```

=====PRINT_BPTREE=====
FrequentPattern Frequency
{almonds, soup} 2
{avocado, soup} 2
{body spray, soup} 2
{chicken, soup} 2
{frozen vegetables, soup} 2
{ground beef, soup} 2
{hot dogs, soup} 2
{milk, soup} 2
{mineral water, soup} 2
{protein bar, soup} 2
{almonds, soup} 2
{avocado, soup} 2
{body spray, soup} 2
{chicken, soup} 2
{frozen vegetables, soup} 2
{ground beef, soup} 2
{hot dogs, soup} 2
{milk, soup} 2
{mineral water, soup} 2
{protein bar, soup} 2
{almonds, burgers, soup} 2
{almonds, eggs, soup} 2
{body spray, green tea, soup} 2
{burgers, eggs, soup} 2
{burgers, french fries, soup} 2
{burgers, green tea, soup} 2
{chocolate, ground beef, soup} 2
{eggs, french fries, soup} 2
{frozen vegetables, soup, spaghetti} 2
{green tea, soup, spaghetti} 2
{milk, soup, spaghetti} 2
{almonds, burgers, eggs, soup} 2
{burgers, french fries, green tea, soup} 2
{almonds, burgers, eggs, soup} 2
{burgers, french fries, green tea, soup} 2
{burgers, soup} 3
{burgers, soup} 3
{chocolate, eggs, soup} 3
{french fries, green tea, soup} 3
{chocolate, eggs, soup} 3
{french fries, green tea, soup} 3
{chocolate, soup} 4
{eggs, soup} 4
{french fries, soup} 4
{soup, spaghetti} 4
{chocolate, soup} 4
{eggs, soup} 4
{french fries, soup} 4
{soup, spaghetti} 4
{green tea, soup} 6
{green tea, soup} 6
=====
{green tea, soup, spaghetti} 2
{milk, soup, spaghetti} 2
{almonds, burgers, soup} 2
{almonds, eggs, soup} 2
{body spray, green tea, soup} 2
{burgers, eggs, soup} 2
{burgers, french fries, soup} 2
{burgers, green tea, soup} 2
{chocolate, ground beef, soup} 2
{eggs, french fries, soup} 2
{frozen vegetables, soup, spaghetti} 2
{green tea, soup, spaghetti} 2
{milk, soup, spaghetti} 2
{almonds, burgers, eggs, soup} 2
{burgers, french fries, green tea, soup} 2
{almonds, burgers, eggs, soup} 2
{burgers, french fries, green tea, soup} 2
{burgers, soup} 3
{burgers, soup} 3
{chocolate, eggs, soup} 3
{french fries, green tea, soup} 3
{chocolate, eggs, soup} 3
{french fries, green tea, soup} 3
{chocolate, soup} 4
{eggs, soup} 4
{french fries, soup} 4
{soup, spaghetti} 4
{chocolate, soup} 4
{eggs, soup} 4
{french fries, soup} 4
{soup, spaghetti} 4
{green tea, soup} 6
{green tea, soup} 6
=====

```

```

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

```

다음은 PRINT_CONFIDENCE 명령어이다. bpTree에 저장된 Frequent Pattern 중 입력된 상품과 연관율 이상의 confidence값을 가지는 Frequent Pattern을 출력하는 명령어이다. 인자가 모두 입력되지 않거나 형식이 다르면 에러코드를 출력한다. 또한 출력할 패턴이 없거나 B+Tree가 비어 있는 경우에도 에러코드를 출력한다. 다음 결과 화면은 eggs 0.2로 출력한 결과화면이다.

```

=====PRINT_CONFIDENCE=====
FrequentPattern Frequency Confidence
{almonds, eggs} 2 0.4
{burgers, eggs} 2 0.4
{chicken, eggs} 2 0.4
{eggs, french fries} 2 0.4
{eggs, mineral water} 2 0.4
{eggs, turkey} 2 0.4
{almonds, eggs} 2 0.4
{burgers, eggs} 2 0.4
{chicken, eggs} 2 0.4
{eggs, french fries} 2 0.4
{eggs, mineral water} 2 0.4
{eggs, turkey} 2 0.4
{almonds, burgers, eggs} 2 0.4
{almonds, eggs, soup} 2 0.4
{burgers, eggs, soup} 2 0.4
{chicken, eggs, mineral water} 2 0.4
{eggs, french fries, soup} 2 0.4
{almonds, burgers, eggs} 2 0.4
{almonds, eggs, soup} 2 0.4
{burgers, eggs, soup} 2 0.4
{chicken, eggs, mineral water} 2 0.4
{eggs, french fries, soup} 2 0.4
{almonds, burgers, eggs, soup} 2 0.4
{almonds, burgers, eggs, soup} 2 0.4
{chocolate, eggs} 3 0.6
{chocolate, eggs} 3 0.6
{chocolate, eggs, soup} 3 0.6
{chocolate, eggs, soup} 3 0.6
{eggs, soup} 4 0.8
{eggs, soup} 4 0.8
=====

```

```

=====PRINT_CONFIDENCE=====
ERROR 600
=====

```

다음은 PRINT_RANGE명령어이다. PRINT_RANGE명령어는 bpTree에 저장된 패턴을 입력 받은 범위만큼 출력하는 명령어이다. 인자가 모두 입력되지 않거나 형식이 다르면 에러코드를 출력한다. 또한, 출력할 패턴이 없거나 bpTree가 비어 있는 경우 에러코드를 출력한다. 다음 결과 화면은 soup 2 5로 출력한 결과화면이다.

```

=====PRINT_RANGE=====
FrequentPattern Frequency
{almonds, soup} 2
{avocado, soup} 2
{body spray, soup} 2
{chicken, soup} 2
{frozen vegetables, soup} 2
{ground beef, soup} 2
{hot dogs, soup} 2
{milk, soup} 2
{mineral water, soup} 2
{protein bar, soup} 2
{almonds, soup} 2
{avocado, soup} 2
{body spray, soup} 2
{chicken, soup} 2
{frozen vegetables, soup} 2
{ground beef, soup} 2
{hot dogs, soup} 2
{milk, soup} 2
{mineral water, soup} 2
{protein bar, soup} 2
{almonds, burgers, soup} 2
{almonds, eggs, soup} 2
{body spray, green tea, soup} 2
{burgers, eggs, soup} 2
{burgers, french fries, soup} 2
{burgers, green tea, soup} 2
{chocolate, ground beef, soup} 2
{eggs, french fries, soup} 2
{green tea, soup, spaghetti} 2
{milk, soup, spaghetti} 2
{almonds, burgers, eggs, soup} 2
{burgers, french fries, green tea, soup} 2
{almonds, burgers, eggs, soup} 2
{burgers, french fries, green tea, soup} 2
{burgers, soup} 3
{burgers, soup} 3
{chocolate, eggs, soup} 3
{french fries, green tea, soup} 3
{chocolate, eggs, soup} 3
{french fries, green tea, soup} 3
{chocolate, soup} 4
{eggs, soup} 4
{french fries, soup} 4
{soup, spaghetti} 4
{chocolate, soup} 4
{eggs, soup} 4
{french fries, soup} 4
{soup, spaghetti} 4

```

```

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

```

마지막으로 모든 메모리를 해제하고, 프로그램을 종료하는 EXIT 명령어의 결과화면이다.

```

=====EXIT=====
Success
=====

```

valgrind를 통해서 메모리 누수를 확인한 결과 모두 해제가 되었음을 확인할 수 있었다.


```

peter@LAPTOP-2UTG8NG5:~/DS_project2_draft/DS_Project_2_2022_2$ valgrind ./run
==3807== Memcheck, a memory error detector
==3807== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3807== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3807== Command: ./run
==3807==
==3807==
==3807== HEAP SUMMARY:
==3807==   in use at exit: 0 bytes in 0 blocks
==3807==   total heap usage: 5,066 allocs, 5,066 frees, 460,335 bytes allocated
==3807==
==3807== All heap blocks were freed -- no leaks are possible
==3807==
==3807== For lists of detected and suppressed errors, rerun with: -s
==3807== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Consideration

우선 이전에 구현했던 Tree들은 자식을 최대 2개까지만 가졌기 때문에 pointer로 자식을 관리하였다. 그러나 본 프로젝트에서는 다중 검색 트리를 구현한다. 따라서 map으로 자식 노드를 관리하다 보니 이전에 구현한 방식과 다르다 보니 어려움이 있었다. 데이터구조 설계 시간에 배운 내용과 실습 자료를 종합하여 포인터의 개념으로 이해하니 map으로 자식이 여러 개인 노드의 트리를 구현할 수 있었다.

다음은 FP-Tree의 출력문제이다. PRINT_FPTREE를 출력하는 명령어를 구현할 때 현재 노드를 출력하려고 할 때 현재 노드는 빈도수만을 가지고 item 이름은 가지고 있지 않다. 이는 노드의 Children을 Map으로 관리하기 때문에 생기는 문제이다. 따라서 현재 노드를 출력하려면 그 노드의 부모 노드로 이동하여 부모 노드의 map을 접근하여 item을 출력하도록 문제를 해결하였다.

다음은 BpTree의 포인터 연결 문제이다. Data 수가 적은 result1으로 BpTree를 구축하니 문제 없이 출력이 되어서 모두 구현을 하였다고 생각을 하였는데, Data 수가 많은 result2로 BpTree를 구축하니 PRINT_RANGE 명령어와 PRINT_BPTREE 명령어에 대한 결과화면이 중간에 노드들 간의 연결이 끊기는 것처럼 Key값이 날라가버리는 문제가 발생하였다. 처음에 세운 가설은 Datanode를 Split하는 과정에서 Doubly linked list 연결이 잘못되었을 것이라는 가설을 세웠다. 이 문제를 해결하기 위해서 일단 BpTree의 전체 Datanode을 출력하는 함수를 구현하였다. 이 함수는 BpTree의 가장 왼쪽 하단 노드로 이동하여 Doubly linked list로 연결된 Datanode를 모두 출력하는 형태로 구현하였다. 처음에 모든 노드를 Insert하고 전체를 출력하는 함수를 호출하니 어느 부분에서 Split을 할 때에 어느 부분에서 연결이 끊기는 지에 대해서 정확한 지점을 찾기가 어려웠다. 그래서 생각한 방법이 Insert를 해줄 때마다 BpTree를 출력하여 어떠한 Split을 해줄 때 문제가 발생하는 지에 대해서 직접 그림을 그려보며 확인하였다. 확인해본 결과 IndexNode의

LeftSplit노드에서 부모의 연결을 올바르게 시켜주지 않아서 생기는 문제였다. LeftSplit 노드는 함수를 구현하면서 새로 만든 노드이기 때문에 모든 값을 재할당해주어야 한다는 점을 간과하여 생긴 문제였다.

마지막으로 매니저 함수에서의 예외처리에 대한 고찰이다. 이전에 프로젝트를 구현할 때에는 if문으로 예외를 처리해주었다. if문으로 예외를 처리하니, 코드의 라인수와 가독성이 매우 떨어졌었다. 그러나, 데이터구조 설계 시간에 배운 try throw catch구문을 이용하여 문제가 생길 것 같은 지점을 try로 묶고 예외에 대해서 throw로 errorcode를 넘겨주니 catch로 errorcode만 출력을 하면 되어서 코드를 훨씬 간결하고 직관적으로 확인할 수 있게 되었다.