DS Project 보고서

2021202054 송예준

1. Introduction

- 요약

이번 프로젝트에서는 FP-Growth와 B+-Tree를 이용하여 상품 추천 프로그램을 구현한다. 이 프로그램은 장바구니 데이터에서 같이 구매한 상품들을 받아 FP-Growth를 구축한다. FP-Growth는 상품들의 연관성을 Tree 구조로 저장하고 있는 FP-Tree와 상품별 빈도수 및 정보, 해당 상품과 연결된 FP-Tree의 상품 노드들을 관리하는 Header Table로 구성된다. FP-Growth 구축 단계에서 연관된 상품들을 묶은 Frequent Pattern들은 SAVE 명령어를 통해 result.txt에 빈도수, 상품 순으로 저장한다. Frequent Pattern들이 저장된 result.txt는 BTLOAD 명령어를 통해 빈도수를 기준으로 B+-Tree에 저장이 된다. B+-Tree는 IndexNode와 DataNode로 구성된다. IndexNode는 DataNode를 찾기 위한 Node이고 DataNode는 해당 빈도수를 가지는 Frequent Pattern들이 저장된 Node이다. 또, 추가로 FP-Tree와 Header Table 생성 및 연결과 B+-Tree이며, FP-Growth의 FrequentPattern들을 찾아서 result.txt에 저장한다.

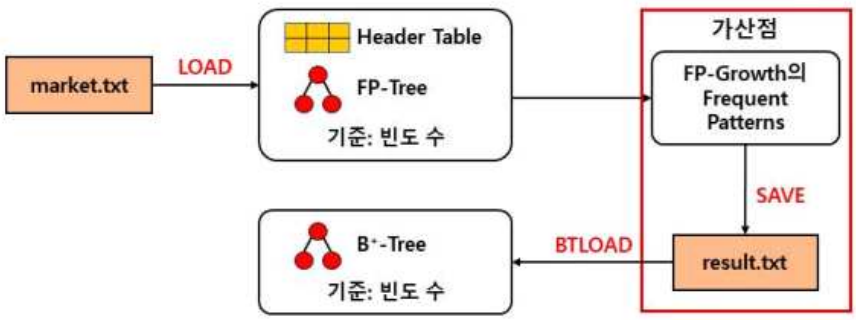


그림 1. 상품 추천 프로그램 구조

1. FP-Growth

- 주어진 market.txt에 저장된 데이터를 이용하여 구축한다. market.txt에서 같이 구매한 물품은 줄 단위로 구분되어 있다.
- 프로그램 구현 시 주어진 FPNode 클래스를 통해 FP-Tree를 구현하고 주어진 Header Table 클래스를 통해 Header Table을 구현한다.
- 상품 정보는 항상 고유하며, 소문자로 표기한다고 가정한다.
- FP-Growth의 threshold 값은 고정되어 있지 않으며 멤버 변수로 변경할 수 있다.
  - threshold 값은 2 이상으로 설정한다고 가정한다.
- Header Table, FP-Tree, B+-Tree의 구축 방법과 조건에 대한 자세한 설명은 아래에서 추가 설명한다.
- FP-Growth가 생성되면 Frequent Pattern들을 result.txt에 저장한다.

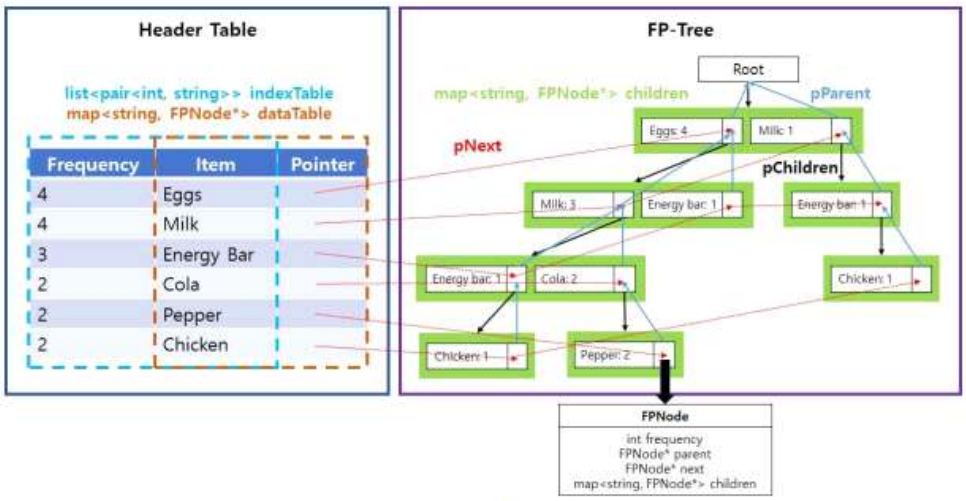


그림 2. FP-Growth의 예

2. Header Table

- Header Table은 인덱스 테이블(indexTable)과 데이터 테이블(dataTable)로 구성한다.
- Header Table에는 threshold보다 작은 상품들도 저장한다.

- 인덱스 테이블은 빈도수를 기준으로 정렬된 상품들을 저장하는 변수다. 이 변수는 아래와 같이 STL의 list를 사용하며 list에는 pair를 통해 빈도수와 상품명을 저장한다.

  • list<pair<int, string>> indexTable

- 인덱스 테이블에서는 빈도수를 기준으로 오름차순과 내림차순으로 정렬을 할 수 있는 함수가 구현되어야 하며, 이는 list에서 제공되는 sort 함수를 이용하여 구현한다.

- 데이터 테이블은 상품명과 상품에 연결되는 포인터를 저장하는 변수다. 이 변수는 아래와 같이 STL의 map 컨테이너 형태로 key와 value로 상품명과 상품에 연결되는 FP-Tree의 Node (FPNode) 포인터를 저장한다.

  • map<string, FPNode*> dataTable

3. FP-Tree

- FP-Tree 클래스는 따로 생성하지 않고 FPNode를 이용하여 구축한다.

- root에서 자식 노드를 제외한 변수들은 NULL 값을 갖는다.

- 자식 노드들은 아래와 같이 map 컨테이너 형태로 저장하며, 부모 노드를 가리키는 노드가 존재한다. key의 string은 상품명을 저장하고 value의 FPNode*는 해당 상품의 빈도수 정보 및 연결된 Node 정보들을 저장한다.

  • map<string, FPNode*> children

- FP-Tree에 저장된 노드들은 Header Table에서 같은 상품 노드들끼리 연결되어야 한다.

- FP-Tree에서는 각각 연결된 연관 상품에 따라 빈도수를 정확히 설정해야 한다. - FP-Tree에 연결되는 상품 순서는 빈도수를 기준으로 내림차순으로 결정이 되며, Header Table의 정렬 기준에 따라 결정이 된다. - 빈도수가 같은 경우 Header Table에서 indexTable의 정렬 기준에 따라 결정이 된다.

- children에 저장되는 데이터는 상품명과 상품 노드 (FPNode)이며 상품

노드에는 빈도수 정보, 부모 노드 정보, 자식 노드 정보가 저장되어야 한다.

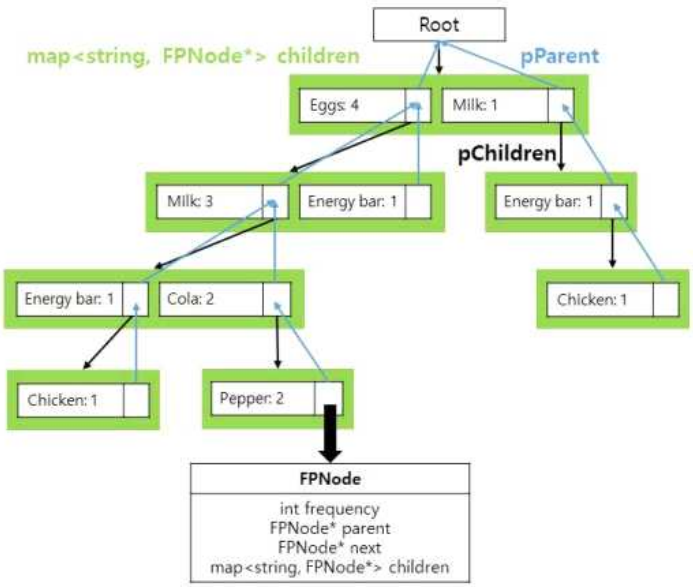


그림 3. FP-Tree의 예

4. B+-Tree

- result.txt에 저장된 데이터를 이용하여 구축한다.
- result.txt는 빈도수가 제일 처음으로 주어지며 그 뒤에 "₩t"을 구분자로 하여 연관된 상품들(Frequent Patterns)을 저장한 파일이다.
- B+-Tree는 그림 4와 같이 빈도수를 기준으로 정렬된다.
- B+-Tree는 인덱스 노드 (BpTreeIndexNode)와 데이터 노드 (BpTreeDataNode)로 구성되며, 각 노드 클래스는 B+-tree 노드 클래스 (BpTreeNode)를 상속받는다.
- 데이터 노드는 단말 노드로, 해당 빈도수에 속하는 Frequent Pattern이 저장된 FrequentPatternNode를 아래와 같이 map 컨테이너 형태로 가지고 있으며 가장 왼쪽 자식을 가리키는 포인터(pMostLeftChild)를 따로 가지고 있다.

  • map<int, FrequentPatternNode*> mapData

- B+-Tree의 차수 ORDER(m)는 고정되어 있지 않으며 멤버 변수로 변경할 수 있다.

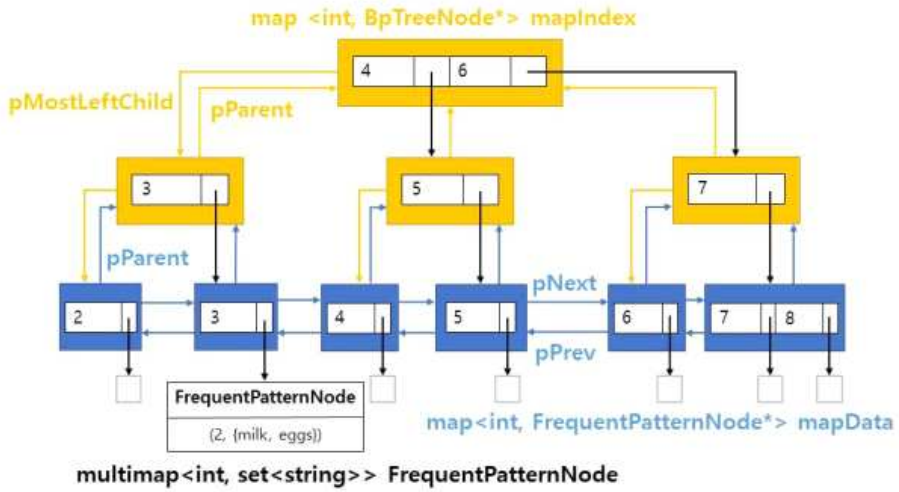


그림 4. B⁺-Tree의 예

5. FrequentPatternNode

- FrequentPatternNode는 Frequent Pattern 정보를 아래와 같이 multimap 컨테이너 형태로 가지고 있다.

  • multimap<int, set<string>> FrequentPatternList

- 각 mapData의 빈도수에 해당하는 Frequent Pattern의 정보를 저장하고 있어야 하며 key와 value로 Frequent Pattern의 길이와 Frequent Pattern의 원소를 저장해야 한다.

- Frequent Pattern의 원소는 아래와 같이 STL의 set 컨테이너 형태로 가지고 있다. set에는 원소에 해당하는 상품명이 저장돼야 한다.

  • set<string> item

- Frequent Pattern 중 공집합 이거나 원소가 하나인 집합은 저장하지 않는다.

- 구현해야 할 명령어

1. Load

   텍스트 파일의 데이터 정보를 불러오는 명령어로, 텍스트 파일에 데이터가 존재하는 경우 텍스트 파일을 읽어 FP-Growth를 생성 한다. 만약 텍스트 파일이 존재하지 않거나 자료구조에 이미 데이터가 들어가 있으면 알맞은 에러 코드를 출력한다. 사용되는 텍스트 파일의 이름은 market.txt이며 이 파일 이름은 수정하지 않는다. (에러코드: 100)

   *데이터 조건
   - 상품명은 무조건 소문자로 주어지며 그 이외의 입력은 없다고 가정한다.
   - "₩t"를 구분자로 하여 상품을 구분한다. - 같이 구매한 물품들은 줄 단위로 구분된다.

2. BTLoad

   텍스트 파일의 데이터 정보를 불러오는 명령어로, 텍스트 파일에 데이터 정보가 존재하는 경우 텍스트 파일을 읽어 B+-Tree에 저장 한다. 만약 텍스트 파일이 존재하지 않거나 자료구조에 이미 데이터가 들어가 있으면 알맞은 에러 코드를 출력한다. 사용되는 텍스트 파일의 이름은 result.txt입니다. (에러코드: 200)

   *데이터 조건
   - 첫 번째 값은 빈도수고 그 이후 값은 상품 정보다.
   - "₩t"를 구분자로 하여 상품을 구분한다.
   - Frequent Pattern에 속한 상품들은 줄 단위로 구분된다.

3. PRINT_ITEMLIST

   FP-Growth의 Header Table에 저장된 상품들을 빈도수를 기준으로 내림차순으로 출력하는 명령어로 threshold보다 작은 빈도수를 가진 상품도 출력한다. Header Table이 비어 있는 경우 에러

코드를 출력한다. (에러코드: 300)

4. PRINT_FPTREE

FP-Growth의 FP-Tree 정보를 출력하는 명령어로 아래 출력 기준에 맞춰 출력해주는 명령어이다. Header Table을 먼저 빈도수를 기준으로 오름차순으로 정렬하고 threshold 이상의 상품들을 출력한다. FP-Tree가 비어 있는 경우 에러 코드를 출력한다. (에러코드: 400)

* 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 root 노드 전까지 연결된 부모 노드를 출력한다.
- 해당 상품과 연결된 다음 노드들이 없을 때까지 출력하며 다음 노드로 이동하면 다음 줄로 이동한다.

5. PRINT_BPTREE

B+-Tree에 저장된 Frequent Pattern 중 입력된 상품과 최소 빈도수 이상의 값을 가지는 Frequent Pattern을 출력하는 명령어이다. 첫 번째 인자로 상품명을 입력받고 두 번째 인자로 최소 빈도수를 받는다. 인자가 부족하거나 형식이 다르면 에러 코드를 출력한다. 명령이 실행되면 입력받은 최소 빈도수를 기준으로 B+-Tree에서 탐색한다. 이후 B+-Tree에 저장된 Frequent Pattern 중 최소 빈도수를 만족하며, 입력된 상품을 포함하는 Frequent Pattern을 출력하며 이동한다. 출력할 Frequent Pattern이 없거나 B+-Tree가 비어 있는 경우 에러코드를 출력한다. (에러코드: 500)

6. PRINT_CONFIDENCE

B+-Tree에 저장된 Frequent Pattern 중 입력된 상품과 연관율

이상의 confidence 값을 가지는 Frequent Pattern을 출력하는 명령어이다. 첫 번째 인자로 상품명을 입력받고 두 번째 인자로 연관율을 받는다. 2개의 인자가 모두 입력되지 않거나 형식이 다르면 에러 코드를 출력한다. (에러코드: 600)

$$\text{연관율} = \frac{\text{부분집합의 빈도수}}{\text{해당 상품의 총 빈도수}}$$

명령이 실행되면 입력받은 연관율과 해당 상품의 총 빈도수의 곱보다 큰 빈도수를 B+-Tree에서 탐색한다. 탐색이 끝나면 B+-Tree에 저장된 연관율 이상의 Frequent Pattern을 출력하며 이동한다. 출력할 Frequent Pattern이 없거나 B+-Tree가 비어 있는 경우 에러 코드를 출력한다. (에러코드: 600)

7. PRINT_RANGE

B+-Tree에 저장된 Frequent Pattern을 출력하는 명령어로, 첫 번째 인자로 상품명을 입력하고 두 번째 인자로 최소 빈도수, 세 번째 인자로 최대 빈도수를 입력받는다. 3개의 인자가 모두 입력되지 않거나 형식이 다르면 에러 코드를 출력한다. 명령이 실행되면 최소 빈도수를 가지고 B+-Tree에서 FrequentPattern을 탐색한다. 탐색이 끝나면 최대 빈도수까지 B+-Tree에 저장된 입력된 상품을 포함한 Frequent Pattern을 출력하며 이동한다. 출력할 Frequent Pattern이 없거나 B+-Tree가 비어 있는 경우 에러 코드를 출력한다. (에러코드: 700)

8. SAVE

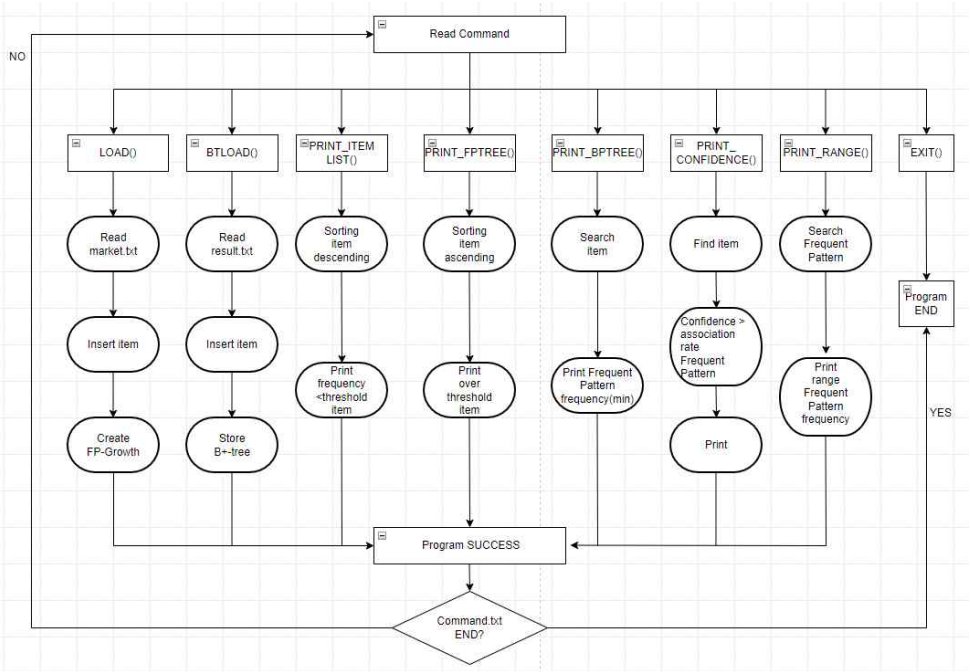
FP-Growth의 상품들의 연관성 결과를 저장하는 명령어로 생성된 Frequent Pattern들을 result.txt에 저장한다. 저장 포맷은 '₩t'를 구분자로 하여 빈도수, 상품명 순으로 저장한다. FP Growth의 Frequent Pattern이 비어 있는 경우 예외 처리를 한다. (에러코드: 800)

9. EXIT

프로그램상의 메모리를 해제하며, 프로그램을 종료한다.

2. **Flowchart** (draw.io 이용)

이번 2차 프로젝트의 전체적인 flowchart는 아래와 같다.



3. **Algorithm**

① Fp-growth 알고리즘(=연관규칙 알고리즘)

Apriori 알고리즘을 조금 더 혁신적으로 개선하여 구현한 알고리즘이다. 일단 이 알고리즘을 적용시키기 위해 가장 먼저 해야 할 일은 Fp-tree를 만드는 것이다. 이 Fp-tree를 만듦으로써 기존에 모두 찾아야했던 frequent itemset들을 Fp-tree 하나에서만 찾을 수 있게 된다.

Market-Basket transactions

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

- 알고리즘 진행 과정

TID	Items Bought	(Ordered) Frequent Items
100	a, b, c, d, e, f, g, h	
200	a, f, g	
300	b, d, e, f, j	
400	a, b, d, i, k	
500	a, b, e, g	

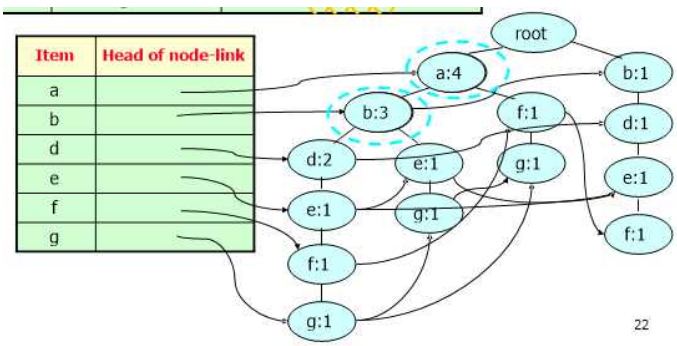
1. 각각의 transcation list를 포함하고 있는 아이템마다 support 값을 계산한다.
2. 구해진 support를 기준으로 support 값이 threshold 이상인 아이템들만 골라낸다.

TID	Items Bought	(Ordered) Frequent Items
100	a, b, c, d, e, f, g, h	a, b, d, e, f, g
200	a, f, g	a, f, g
300	b, d, e, f, j	b, d, e, f
400	a, b, d, i, k	a, b, d
500	a, b, e, g	a, b, e, g

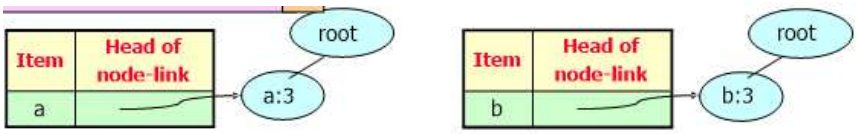
Item	Frequency
a	4
b	4
c	1
d	3
e	3
f	3
g	3
h	1
i	1
j	1
k	1

Item	Frequency
a	4
b	4
d	3
e	3
f	3
g	3

3. 골라낸 아이템들을 support 기준으로 내림차순 정렬하여 header table을 구현한다.
4. transcation마다 root부터 insert하고, 새로운 item들이면 새로운 node를 가르키도록 한다.



5. table에서 bottom up 방법으로 item을 하나씩 prefix로 잡은 후 그 아이템을 포함하는 transcation의 다른 item들의 support 값을 계산하고 thresold보다 크거나 같다면 위와 같은 subtree를 구현한다.
6. 아래와 같은 방법으로 tree를 구현한 후 support 값을 구해 가면서 threshold 이상의 pattern 뽑아낸다. 그런 다음 최종 frequent patterns를 결정한다.



※ 참고 자료 출처

⑴. https://process-mining.tistory.com/92

⑵. 07 FP-Growth Algorithm 강의자료

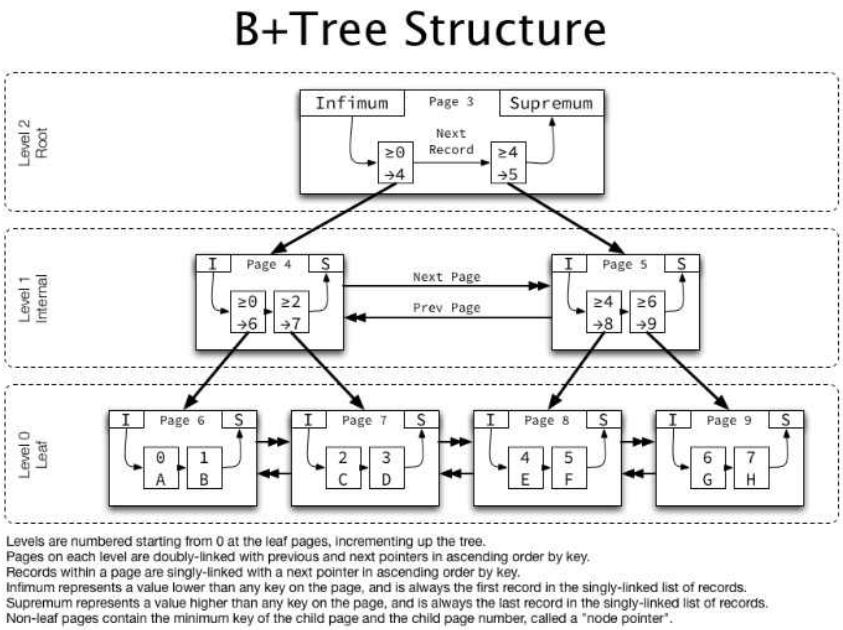
② B+tree 알고리즘

B-tree와 다르게 같은 레벨의 모든 키값들이 정렬되어 있고, 같은 레벨의 Sibiling node는 연결리스트 형태로 이어져 있다.
(같은 레벨의 Sibiling node는 모두 연결되어 있어서 키값이 중복 되지 않는다!)

만약 특정 값을 찾아야 하는 상황이 된다면 leaf node에 모든 자료들이 존재하고, 그 자료들이 연결리스트로 연결되어 있으므로 탐색이 쉬워진다. leaf node가 아닌 자료는 index node, leaf node 자료는 data node라고 한다. index node의 경우 value 값에 다음 node를 가리킬 수 있는 포인터 주소가 존재하고 data node의 value 값에 data가 존재한다.

- 특징

1. 데이터 노드의 자료는 정렬되어 있다.
2. 데이터 노드에서는 데이터가 중복되지 않는다.
3. 모든 leaf node는 같은 레벨에 있다.
4. leaf node가 아닌 node의 키값의 수는 그 노드의 서브트리의 수보다 하나 적다.
5. 모든 leaf node는 linked list로 연결되어 있다.



※ 참고 자료 출처

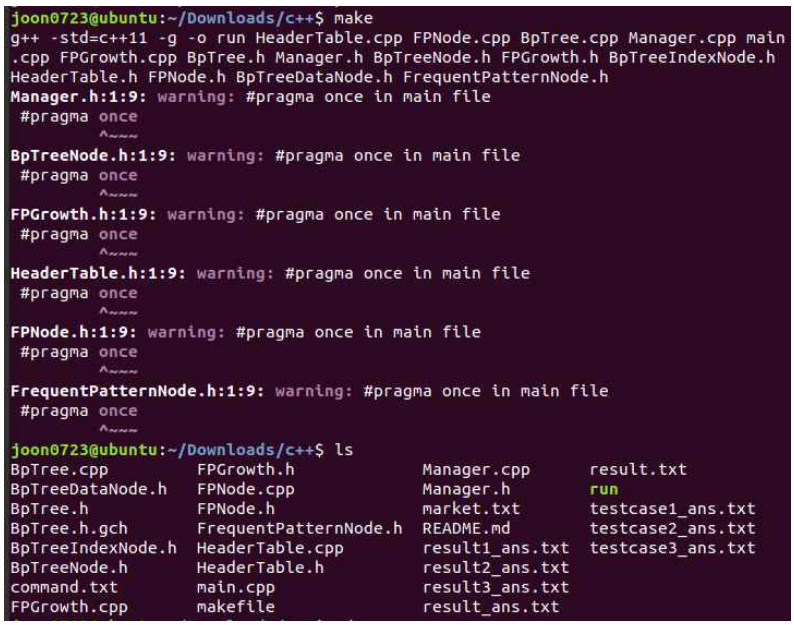
(1). https://ssocoit.tistory.com/217

(2). https://blog.jcole.us/2013/01/10/btree-index-structures-in-innodb/

(순서도 그림)

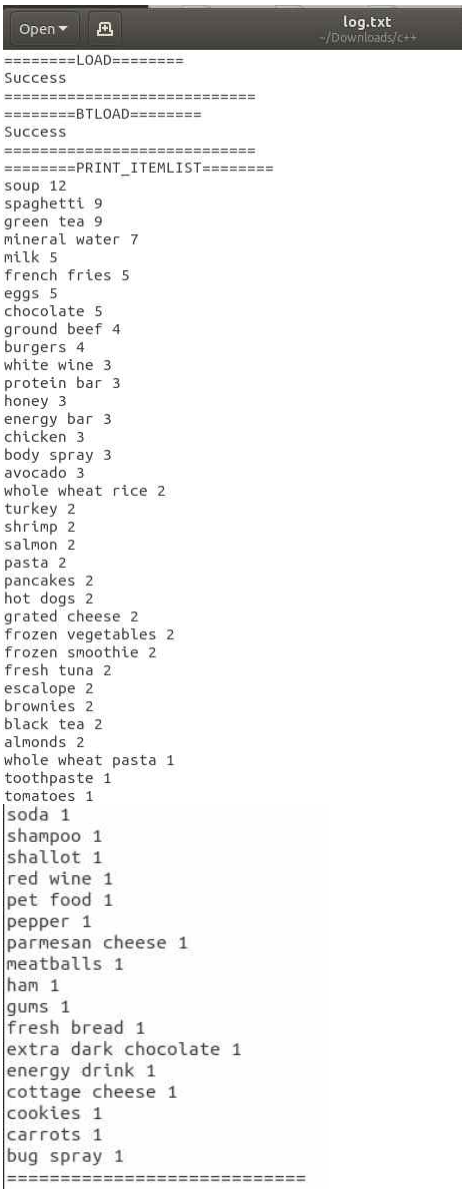
4. Result Screen

- Terminal 창과 log.txt 비교



2차 프로젝트 컴파일 과정은 위와 같다~

- Run Terminal 창과 log.txt 비교



result1과 testcase1 텍스트를 result와 market 텍스트에 삽입 시킨 결과 (LOAD, BTLOAD, PRINT_ITEMLIST)

```
========PRINT_FPTREE========
{StandardItem,Frequency} {Path_Item,Frequency}
{almonds,2}

{black tea,2}

{brownies,2}

{escalope,2}

{fresh tuna,2}

{frozen smoothie,2}

{frozen vegetables,2}

{grated cheese,2}

{hot dogs,2}

{pancakes,2}

{pasta,2}

{salmon,2}

{shrimp,2}

{turkey,2}

{whole wheat rice,2}

{avocado,3}

{body spray,3}

{chicken,3}

{energy bar,3}

{honey,3}

{protein bar,3}

{white wine,3}

{burgers,4}

{ground beef,4}

{chocolate,5}

{eggs,5}

{french fries,5}

{milk,5}

{mineral water,7}

{green tea,9}

{spaghetti,9}

{soup,12}

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

```
========PRINT_FPTREE========
{almonds,2}
{black tea,2}
{brownies,2}
{escalope,2}
{fresh tuna,2}
{frozen smoothie,2}
{frozen vegetables,2}
{grated cheese,2}
{hot dogs,2}
{pancakes,2}
{pasta,2}
{salmon,2}
{shrimp,2}
{turkey,2}
{whole wheat rice,2}
{avocado,3}
{body spray,3}
{chicken,3}
{energy bar,3}
{honey,3}
{protein bar,3}
{white wine,3}
{burgers,4}
{ground beef,4}
{chocolate,5}
{eggs,5}
{french fries,5}
{milk,5}
{mineral water,7}
{green tea,9}
{spaghetti,9}
{soup,12}
============================
```

result1과 testcase1 텍스트를 result 와 market 텍스트에 삽입시킨 결과

(PRINT_FPTREE)

```
========PRINT_BPTREE========
{almonds, burgers, eggs} 3
{almonds, eggs, soup} 3
{burgers, eggs, soup} 3
{chicken, eggs, mineral water} 3
{eggs, french fries, soup} 3
{almonds, burgers, eggs, soup} 4
{chocolate, eggs, soup} 3
============================
========PRINT_CONFIDENCE========
{energy bar, milk} 2 0.25
{milk, mineral water} 2 0.25
{milk, soup} 2 0.25
{milk, spaghetti} 2 0.25
============================
========PRINT_RANGE========
FrequentPattern      Frequency
{energy bar, milk} 2
{french fries, milk} 2
{green tea, milk} 2
{milk, mineral water} 2
{milk, soup} 2
{energy bar, milk, mineral water} 3
{milk, soup, spaghetti} 3
{milk, spaghetti} 2
============================
========SAVE========
Success
============================
========EXIT========
Success
============================
```

```
========PRINT_BPTREE========
{almonds, burgers, eggs} 3
{almonds, eggs, soup} 3
{burgers, eggs, soup} 3
{chicken, eggs, mineral water} 3
{eggs, french fries, soup} 3
{almonds, burgers, eggs, soup} 4
{chocolate, eggs, soup} 3
============================
========PRINT_CONFIDENCE========
{energy bar, milk} 2 0.25
{milk, mineral water} 2 0.25
{milk, soup} 2 0.25
{milk, spaghetti} 2 0.25
============================
========PRINT_RANGE========
FrequentPattern      Frequency
{energy bar, milk} 2
{french fries, milk} 2
{green tea, milk} 2
{milk, mineral water} 2
{milk, soup} 2
{energy bar, milk, mineral water} 3
{milk, soup, spaghetti} 3
{milk, spaghetti} 2
============================
========SAVE========
Success
============================
========EXIT========
Success
============================
```

result1과 testcase1 텍스트를 result와 market 텍스트에 삽입 시킨 결과 (PRINT_BPTREE, PRINT_CONFIDENCE, PRINT_RANGE, SAVE, EXIT)

5. Consideration

이번 과제에서 힘들었던 것은 run 실행 과정이었다. run을 우분투로 실행시킨 결과 아래와 같은 창이 계속 나와서 매우 당황스러웠다.

```
joon0723@ubuntu:~/Downloads/c++$ ./run
Segmentation fault (core dumped)
joon0723@ubuntu:~/Downloads/c++$ cat log.txt
========LOAD========
========LOAD========
========LOAD========
joon0723@ubuntu:~/Downloads/c++$
```

Segmentation fault의 원인을 찾아보니 케이스가 너무 많아서 한 번에 문제를 해결하기 매우 어려웠다. 그리고 처음에는 change

directory 과정 중 make 컴파일이 잘 안되서 당황스러웠는데 파일의 위치 설정이 틀려서 몇 시간동안 고민했었다.
(초기 출력 결과 -> make: *** no tragets...)

그리고, 실습 초기에 log 텍스트에서 출력 결과가 몇 번 정도 반복되서 나왔는데 log.txt를 삭제하고 다시 실행시켰더니 원하는 결과가 출력되서 조금 놀라웠다. 또, 콘솔 출력과 log 텍스트 출력과 일치하지 않아서 조금 당황했었는데 금방 해결하였다.(아래 결과)

- 초기 출력 결과 (처음에는 이 결과가 몇 번 반복해서 출력됨)

```
========LOAD========
Success
========BTLOAD========
Success
========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_FPTREE========
{almonds,2}
{black tea,2}
{brownies,2}
{escalope,2}
{fresh tuna,2}
{frozen smoothie,2}
{frozen vegetables,2}
{grated cheese,2}
{hot dogs,2}
{pancakes,2}
{pasta,2}
{salmon,2}
{shrimp,2}
{turkey,2}
{whole wheat rice,2}
{avocado,3}
{body spray,3}
{chicken,3}
{energy bar,3}

{honey,3}
{protein bar,3}
{white wine,3}
{burgers,4}
{ground beef,4}
{chocolate,5}
{eggs,5}
{french fries,5}
{milk,5}
{mineral water,7}
{green tea,9}
{spaghetti,9}
{soup,12}
==========
========PRINT_BPTREE========
==========
========PRINT_CONFIDENCE========
==========
========PRINT_RANGE========
==========
========SAVE========
Success
==========
========EXIT========
Success
==========

명령어 구현하는 과정 중 가장 어려웠던 건 B+tree 관련 명령어 구현이었다. B+tree 구조를 이해하는 과정에서 시간이 굉장히 오래 걸렸고 여러 차례 시행착오를 겪었다. 그러나 긴 수정 끝에 원하는 결론을 낼 수 있었다. 그리고 FP-growth 알고리즘 구현 과정에서 고려해서 사용할 c++ 기능들이 많았고 익숙하지 않아서 시행착오를 살짝 겪었다.

이번 실습에서 어려움과 다르게 살짝 고민된 부분이 있었는데 바로 Command.txt 작성이다. Command 텍스트를 문제에 제시된 예시 그대로 작성해야 되는 것인지 아니면 다르게 해야 되는 것인지 잘 몰라서 처음에는 많이 고민됐다. 그래서 고민 끝에 문제 출력 예시 그대로 하여 log 텍스트 출력 결과를 확인하였다.

그래서 이번 2차 프로젝트를 진행하면서 느꼈던 것은 코드 수정은 1차 프로젝트보다 적었지만 문제에 적용시킬 알고리즘 개념이 어려워서 명확히 알고 따로 더 제대로 공부해야겠다는 생각이 들었다.