PROJECT 2

과목명	데이터구조실습
담당교수	이형근 교수님
학과	컴퓨터정보공학부
학년	2 학년
학번	2019202009
이름	서여지
제출일	2020.11.06 (금)

KWANGWOON UNIVERSITY

1. Introduction

이번 프로젝트는 장바구니 데이터를 이용하여 상품 추천 프로그램을 구현하는 것이다. 프로그램은 장바구니 데이터를 바탕으로 FP-Growth 와 B+ -Tree 를 생성하고, 이것을 이용하여 특정 상품과 함께 구매한 빈도가 높은 상품을 확인할 수 있다. command.txt 에 적힌 명령어에 따라 market.txt 와 result.txt 의 정보를 읽어 처리한 뒤, result.txt 와 log.txt 파일에 결과를 출력한다. 프로그램이 수행할 수 있는 명령어와 각 명령어의 동작은 다음과 같다.

LOAD	market.txt 의 정보를 읽어 FP-Growth 를 생성한다.
BTLOAD	result.txt 의 정보를 읽어 B+ -Tree 에 저장한다.
PRINT_ITEMLIST	FP-Growth 의 Header Table 의 내용을 내림차순으로 출력한다.
PRINT_FPTREE	FP-Growth 의 FP-Tree 의 정보를 출력한다.
PRINT_MIN	B+ -Tree 의 내용을 최소 빈도수 기준으로 출력한다.
PRINT_CONFIDENCE	B+ -Tree 의 내용 중 지정한 연관율 이상인 것만 출력한다.
PRINT_RANGE	B+ -Tree 의 중 특정 구간의 빈도수를 가지는 내용을 출력한다.
SAVE	Frequent Pattern 을 result.txt 에 저장한다.
EXIT	프로그램을 종료한다.

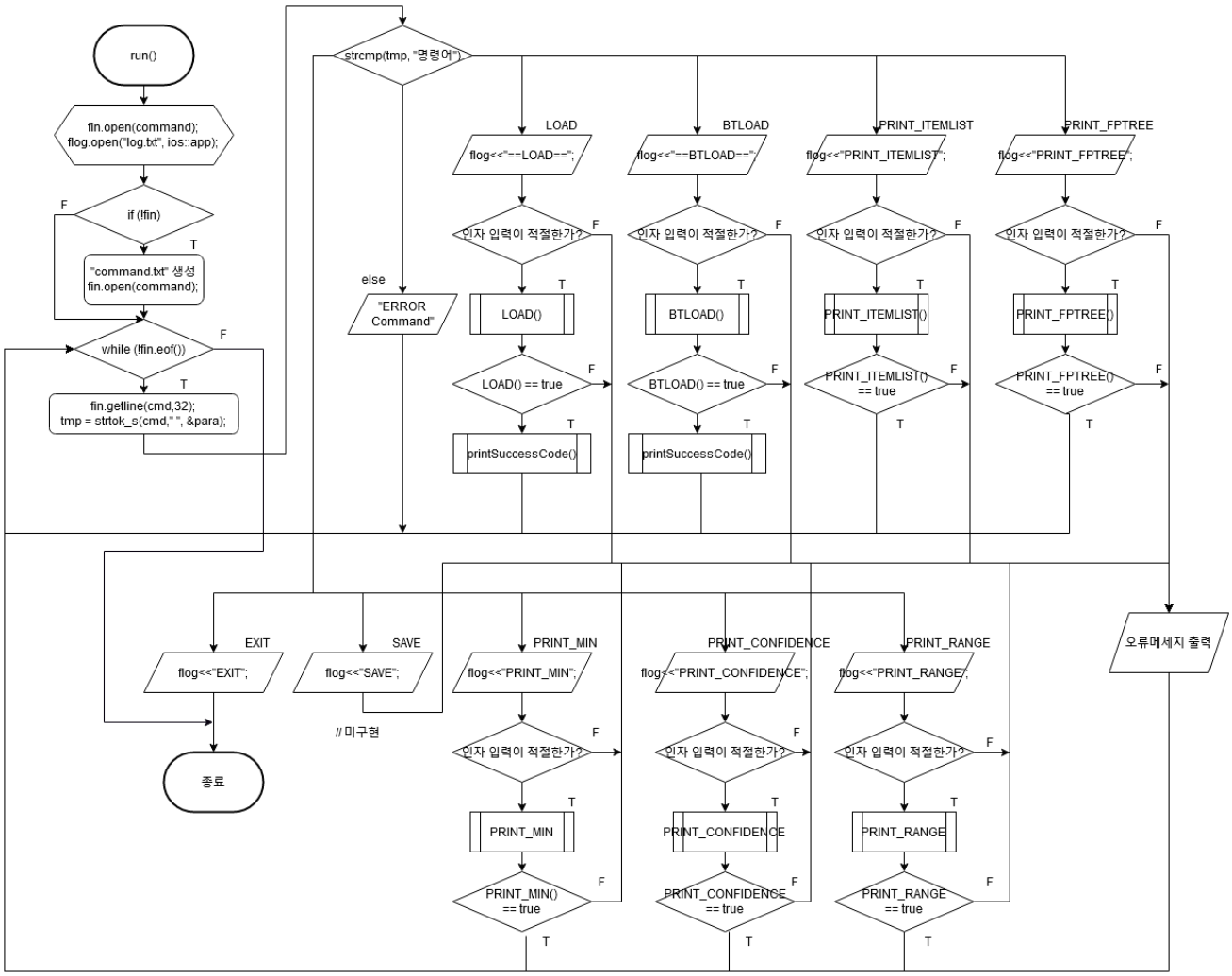
LOAD 동작을 통해 생성되는 FP-Growth 에는 두 가지 자료형이 존재한다. 하나는 indextable 과 datatable 로 구성된 Header Table 이고, 다른 하나는 장바구니의 각 구매 내역에 따라 생성된 FP-Tree 이다. Header Table 은 market.txt 의 자료를 한 줄씩 읽어 '₩t' 문자를 기준으로 각각의 item 을 index table 에 삽입하고, data table 에도 해당 item 을 추가한다. 이미 header table 에 item 이 존재하는 경우에는 해당 item 의 빈도수를 1 증가시킨다. FP-Tree 는 다시 market.txt 의 자료를 한 줄씩 읽어 각 줄에 위치한 item 을 모두 list 에 삽입한다. list 에 삽입된 item 을 내림차순으로 정렬된 index table 의 순서대로 정렬하고, createFPtree 함수에서 FP-TREE 를 생성한다.

BTLOAD 를 통해 result.txt 의 정보를 B+ -Tree 에 저장한다. result.txt 를 한 줄씩 읽어 각 줄에 대해 호출한 Insert 함수에서 B+ -Tree 를 생성한다. B+ -Tree 가 존재하지 않으면 새로 생성하고, 동일한 빈도수를 갖고있는 FrequentPatternNode 가 존재하면 해당 node 에 내용을 삽입한다. 동일한 빈도수를 가진 FrequentPatternNode 가 없는 겨우 새로 생성하고 빈도수에 해당하는 dataNode 에 삽입한다. 이때 dataNode 에서 split 이 일어날 수 있다. 더 이상 split 이 일어나지않을 때 까지 dataNode 와 indexNode 를 분할하는 동작을 한다.

2. Flowchart: 설계한 프로젝트의 플로우차트

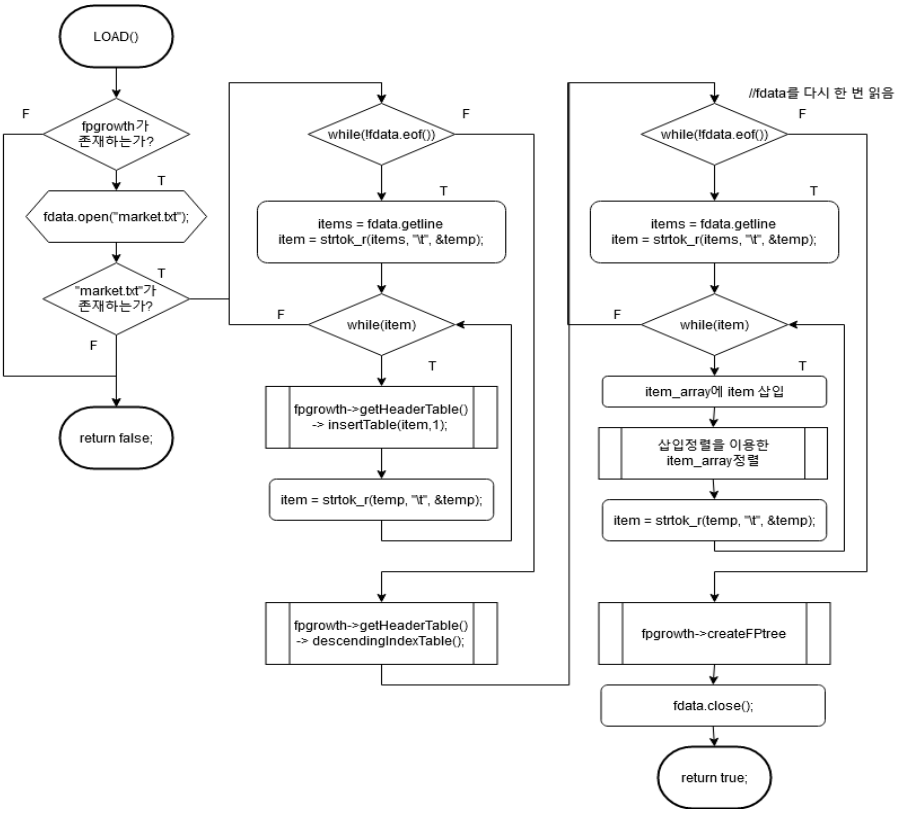
(1) 전체 흐름도 - Manager.run()

프로그램의 전체 기능을 연결하는 run 함수의 순서도이다. command.txt, log.txt 파일을 열어 사용한다. command.txt 파일의 명령어를 읽고, 해당하는 함수를 호출한다. command.txt 가 존재하지 않는 경우, 새로 생성 후 동작한다.



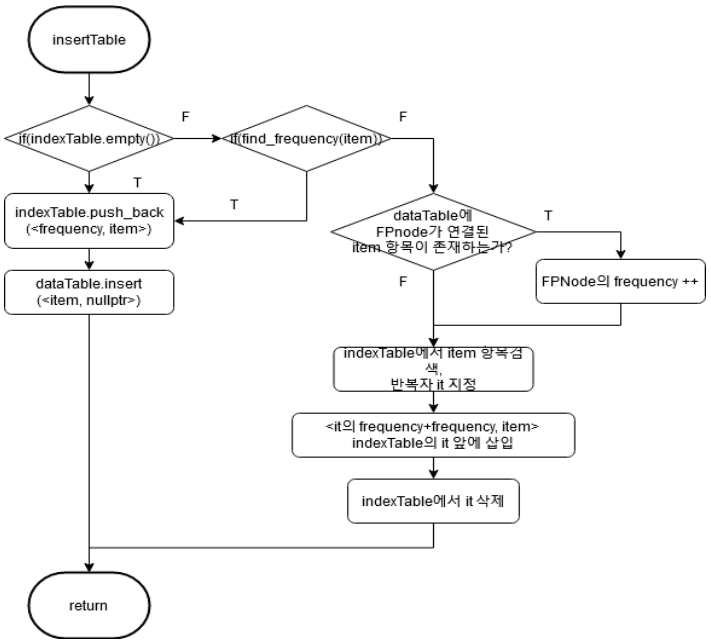
(2) LOAD

market.txt 파일을 읽어 headertable 과 fp tree 를 생성한다. 파일을 한 줄씩 읽어 headertable 을 작성하고, 다시 market.txt 를 처음부터 읽어서 fp tree 를 생성한다.



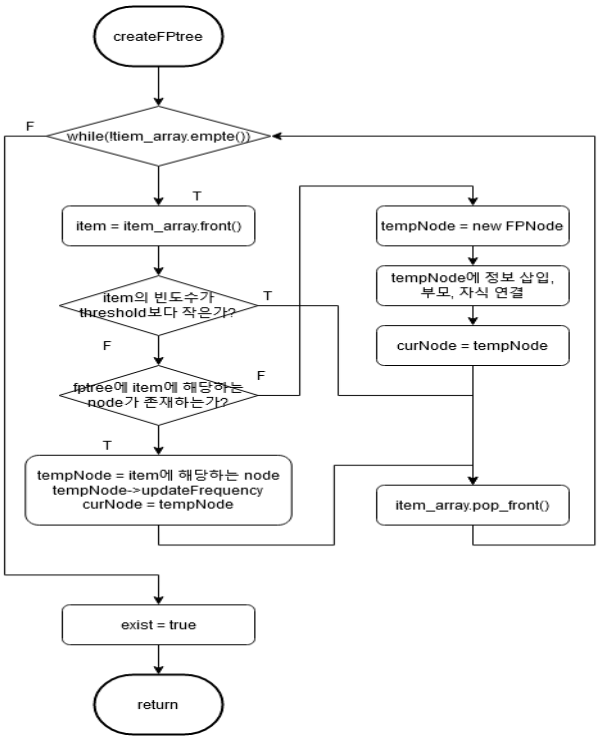
A. insertTable

headertable 에 정보를 삽입하는 함수이다. 정보를 저장할 공간이 있는지 확인 후에 없다면 새로 생성하여 삽입하고, 있다면 frequency 를 증가시킨다.



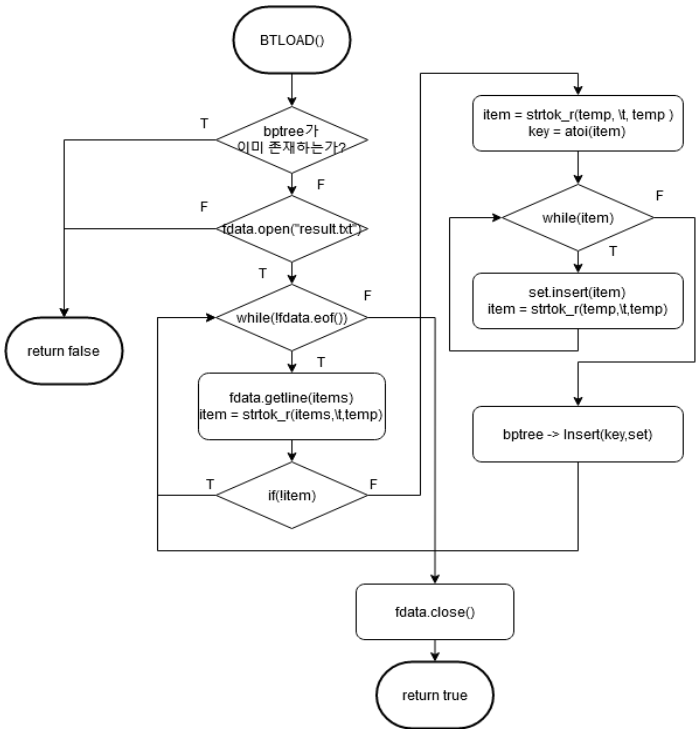
B. createFPtree

FPtree 를 생성하는 함수이다. item_array 를 입력받아 FPtree 에서 내부 요소인 item 에 해당하는 node 를 찾고, 없으면 새로 생성해서 삽입한다. 이미 node 가 있으면 updataFrequency 함수를 이용하여 빈도수를 증가시킨다.



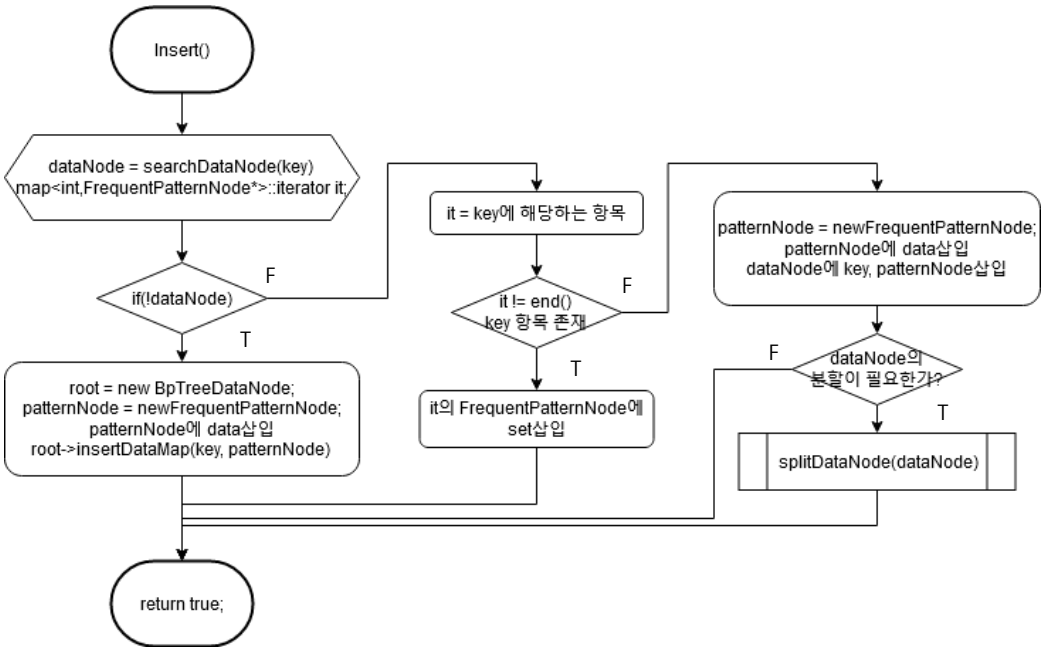
(3) BTLOAD

result.txt 의 정보를 읽어 bptree 를 생성하는 함수이다. BTLOAD 함수에서는 인자로 전달된 key 와 item 을 분리하고 Insert 함수를 호출한다.



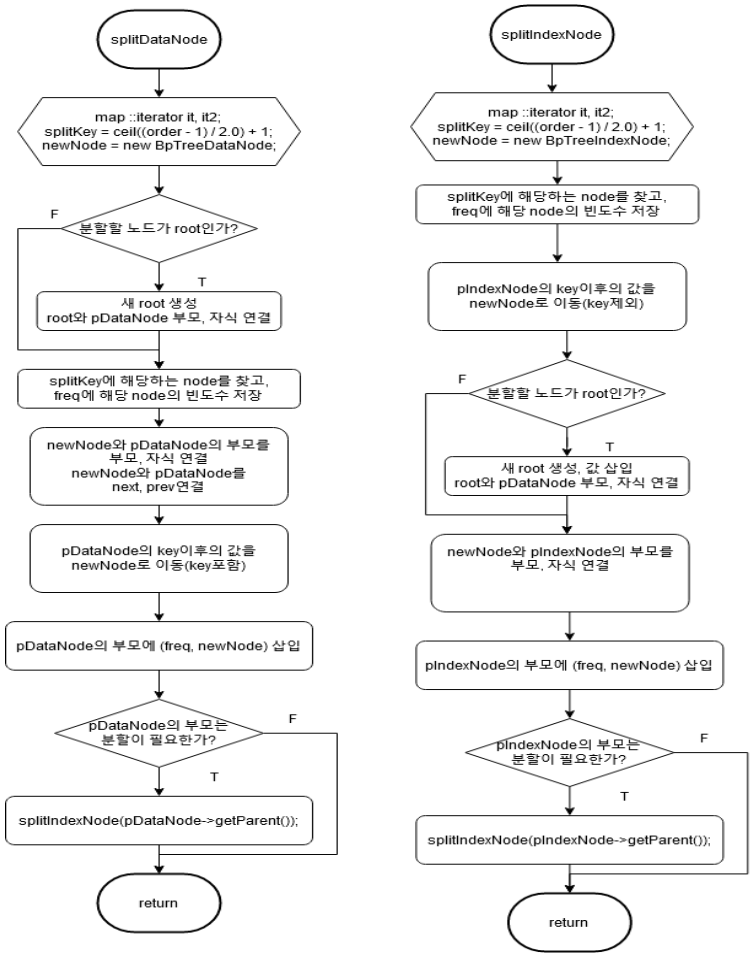
A. Insert

Insert 함수는 bptree 에 직접 data 를 삽입하는 동작을 한다. root node 인 dataNode 를 분리하는 경우, 이미 key 에 해당하는 항목이 존재하는 경우, 존재하지 않는 경우로 나누어 삽입한다. key 에 해당하는 항목이 존재하지 않아 새로 생성 후 삽입하는 경우 dataNode 의 분할이 일어날 수 있다. 분할은 splitDataNode 함수를 호출하여 진행된다.



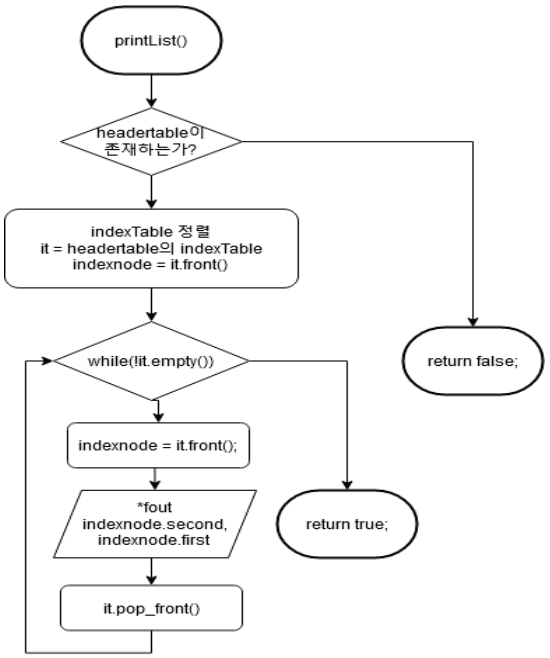
B. splitDataNode & splitIndexNode

DataNode 혹은 IndexNode 의 분리를 실행하는 함수이다. 두 종류의 분할동작은 유사하다. dataNode 의 경우 새 노드를 생성하여 값을 복사했을 때 split 의 기준이 된 항목까지 포함한다는 것이 해당 항목을 갖지 않는 IndexNode 와 다르다.



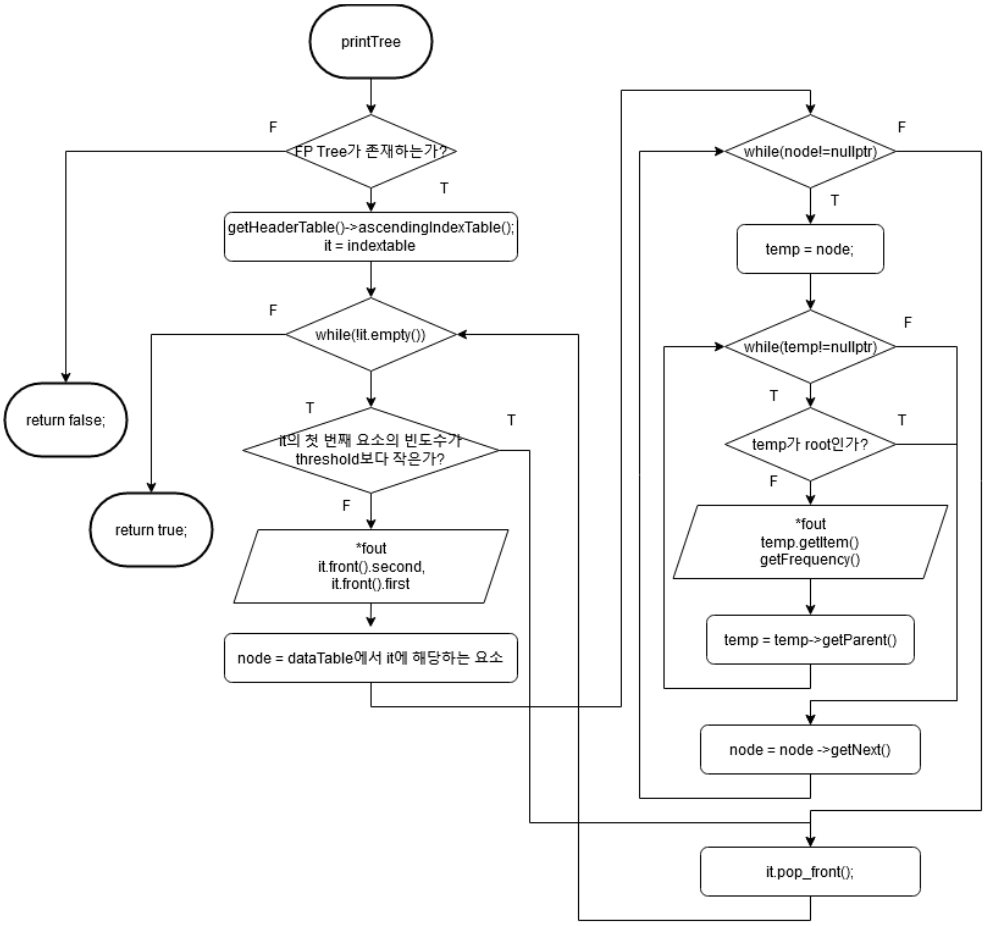
(4) PRINT_ITEMLIST

headertable 을 출력하는 함수이다. headertable 이 존재하지 않는 경우 false 를 반환하고, 존재하는 경우 내림차순으로 정렬하여 모든 항목에 대해 출력한다.



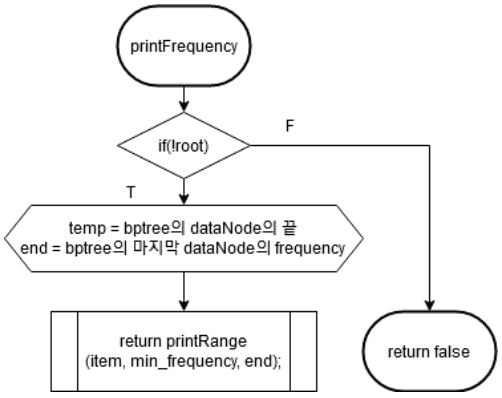
(5) PRINT_FPTREE

fptree 를 출력하는 함수이다. item 의 순서는 headertable 의 오름차순이고, dataTable 에 저장된 순서대로 방문한다. 한 번의 방문에서 부모노드로 거슬러올라가 root 이전까지 한 줄에 출력한다.



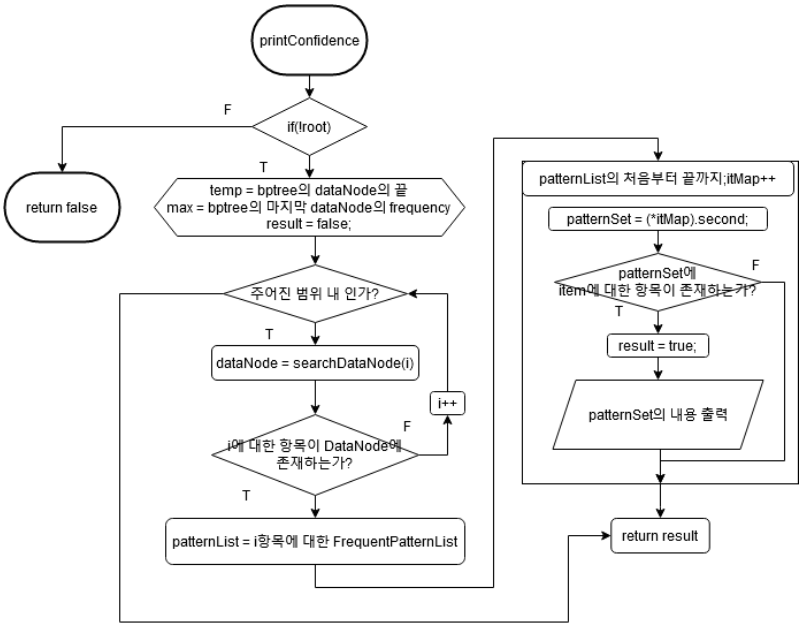
(6) PRINT_MIN

최소 빈도수를 입력 받아 bptree 에서 더 높은 빈도수를 가지고 있는 항목을 출력하는 함수이다. 비슷한 동작을 하는 printRange 함수를 이용하여 구현하였다. 해당 함수를 이용하기 위해 마지막 주파수를 구하는 과정을 거친다.



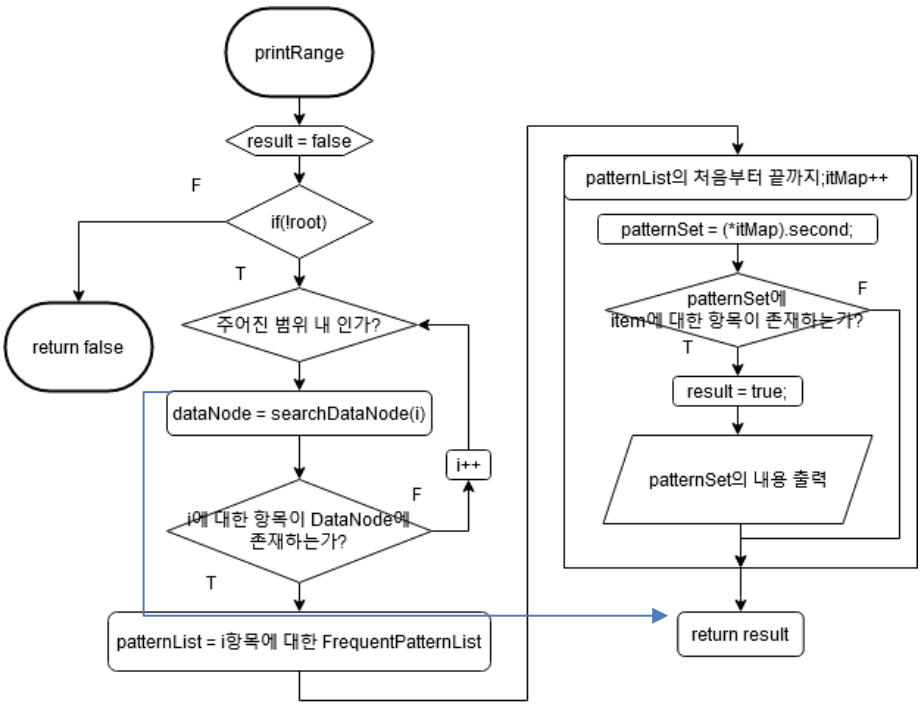
(7) PRINT_CONFIDENCE

연관율을 입력받아 해당 연관율 이상의 값을 갖는 항목을 출력하는 함수이다. 다른 bptree 의 출력함수인 printFrequency 와 printRange 함수를 이용하여 작성하였다.



(8) PRINT_RANGE

bptree 에서 특정 구간의 빈도수에 해당하는 항목을 모두 출력하는 함수이다. bptree 의 dataNode 가 서로 연결되있는 점을 이용하여 작성하였다.



3. Algorithm: 프로젝트에서 사용한 알고리즘

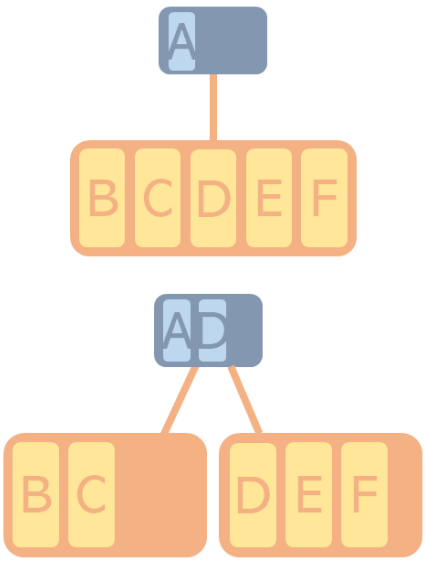
(1) 삽입정렬 알고리즘

market.txt 를 읽어 fpgrowth 를 생성하는 과정에서 fp tree 를 생성하기 위해서 이용되는 list 는 int 와 string 을 갖는 pair 를 요소로 갖는다. (list<pair<int,string>>) fp tree 의 생성에서 순서의 차이로 인해 같은 경로가 다르게 나타나는 것을 막기 위해서는 인자로 전달되는 list 가 pair 내부의 int 를 기준으로 정렬되어있어야 한다. list 를 정렬하기 위해서 삽입정렬 알고리즘을 이용하였다. 삽입정렬 알고리즘은 삽입할 요소에 대해 배열의 앞에서부터 크기를 비교하여 적절한 자리에 요소를 삽입하는 구조이다. 작성한 코드에서는 while 반복문을 통해 item_array 가 빌 때까지 모든 요소에 대해서 insertIter 를 이용한 for 반복문 내부에서 처음부터 항목을 비교한 후 적절한 자리에 요소를 삽입하도록 구현되었다. list 는 insert 를 이용하여 중간에 삽입하는 것이 가능하기 때문에 list 에 pair 를 하나씩 삽입하며 정렬하기에 좋은 알고리즘이라고 생각하여 삽입정렬을 선택하였다.

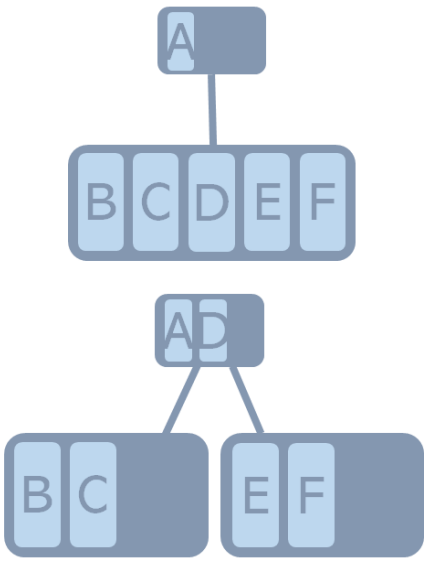
(2) 노드 분할 알고리즘

BpTree 의 생성과정에서 dataNode 에 삽입된 항목이 지정된 order 를 초과할 경우 해당 노드를 분할하여 bptree 를 생성하게된다. node 의 분할은 data node 와 index node 가 다르게 실행된다. data node 는 tree 의 leaf 를 구성하는 node 이고, tree 의 모든 정보를 포함하고 있어야 하기 때문에 노드가 분리될 때 상위의 index node 로 삽입되는 요소까지 새로 생성된 data Node 에 저장된다. 반면 index node 는 상위 index node 로 삽입되는 요소는 새로 생성된 index node 에 저장되지 않는 차이가 있다.

dataNode 분할

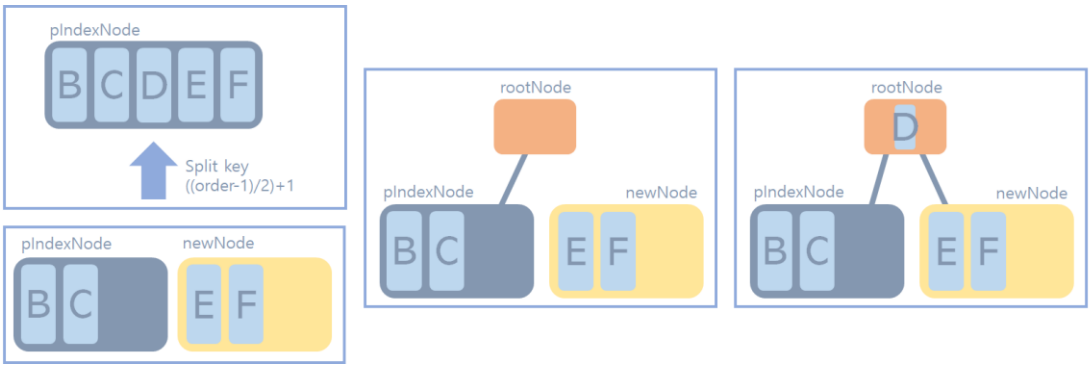


indexNode 분할



위의 차이점을 제외하면 두 종류의 노드가 분할되는 과정은 유사하다. 따라서 index node 의 경우에 대해 알고리즘을 설명한다. node 는 ((order-1)/2)+ 1 번째 요소를 기준으로 분할된다. 분할할 노드인 pIndexNode 의 기준요소 이후의 모든 요소를 분할에 의해 생성될 새 노드 newNode 로 이동시킨다. dataNode 의 경우 기준요소를 포함하여 이동시킨다. pIndexNode 가 root 인 경우 새로운 root node 를 생성하여 기존의 pIndexNode 와 부모, 자식 연결을 실행한다. 새로 생성된 newNode 의 부모는 pIndexNode 와 같기 때문에 pIndexNode 의 부모노드와 newNode 를 부모, 자식 관계로 지정한다. 지정된 pIndexNode 의 부모에 분할의 기준이 된 요소를 삽입한다. 이때 pIndexNode 의 부모노드가 또다시 분할이 필요하다면 다시 분할 함수를 호출하여 분할과정을 진행한다. 이 재귀호출은 더 이상 분할된 함수의 부모노드가 다시 분할 되지 않을 때까지 반복된다.

root 인 IndexNode 를 분할하는 과정



4. Result Screen: 모든 명령어에 대한 결과화면

(1) LOAD

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

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

(2) PRINT_ITEMLIST

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

(3) PRINT_FPTREE

```
======== PRINT_FPTREE ========
{StandardItem,Frequency} (Path_Item,Frequency)
{almonds,2}
(almonds,1) (burgers,1) (french fries,1) (eggs,1) (green tea,4) (soup,12)
(almonds,1) (turkey,1) (ground beef,1) (burgers,1) (eggs,1) (chocolate,1) (soup,12)
{black tea,2}
(black tea,1) (fresh tuna,1) (turkey,1) (chicken,1) (eggs,1) (mineral water,3) (spaghetti,5)
(black tea,1) (energy bar,1) (ground beef,1) (milk,1) (mineral water,3) (spaghetti,5)
{brownies,2}
(brownies,1) (hot dogs,1) (body spray,1) (avocado,1) (french fries,1) (green tea,4) (soup,12)
(brownies,1) (white wine,1) (chocolate,1) (green tea,2) (spaghetti,5)
{escalope,2}
```

(4) BTLOAD

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

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

(5) PRINT_MIN

```
======== PRINT_MIN ========
======== ERROR 500 ========
=======================

======== PRINT_MIN ========
StandardItem    FrequentPatternFrequency
soup -> {almonds} 2
soup -> {avocado} 2
soup -> {body spray} 2
soup -> {chicken} 2
soup -> {frozen vegetables} 2
soup -> {ground beef} 2
soup -> {hot dogs} 2
soup -> {milk} 2
soup -> {mineral water} 2
```

(6) PRINT_CONFIDENCE

```
======== PRINT_CONFIDENCE ========
StandardItem    FrequentPattern Confidence
soup -> {chocolate} 4 0.33
soup -> {eggs} 4 0.33
soup -> {french fries} 4 0.33
soup -> {spaghetti} 4 0.33
soup -> {green tea} 6 0.5
=======================
```

(7) PRINT_RANGE

```
======== PRINT_RANGE ========
StandardItem    FrequentPatternFrequency
soup -> {burgers} 3
soup -> {chocolate, eggs} 3
soup -> {french fries, green tea} 3
soup -> {chocolate} 4
soup -> {eggs} 4
soup -> {french fries} 4
soup -> {spaghetti} 4
=======================
```

(9) 명령어 오류(존재하지 않는 명령어)

```
============================
======== ERROR Command ========
============================
```

(9) EXIT

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

5. Consideration: 고찰

이번 과제를 구현하는데 사용한 자료구조는 FP Growth 의 Header Table 과 FP-Tree, B+-Tree 이다. 각 자료형은 각각의 node 에 STL 컨테이너를 포함하고 있다. Tree 의 node 가 갖고있는 list 혹은 map 자료구조에 컨테이너가 포함되고, 컨테이너 내부에는 자료와 노드를 포함한 또다른 컨테이너가 포함되는 구조가 복잡하게 느껴져서 과제 작성에 어려움을 겪었다. 또한 iteratior 를 이용하여 list 와 map 내부의 요소를 다루는

것이 처음이라 여러 번의 시행착오가 있었다. 이번 과제를 해결하며 stl 의 사용법을 다시 한 번 기억하고, iteratior 의 사용방법을 알게되었다. FrequentPattern 을 추출하는 부분의 구현에 실패해서 SAVE 기능을 작성하지 못한 점이 아쉽다.