본 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. 채점 기준은 FP-Tree와 Header Table 생성 및 연결과 B+-Tree이며, FP-Growth의 Frequent Pattern들을 찾아 result.txt 파일로 저장하는 것은 가산점으로 부여된다.
자료구조의 구축 방법과 조건에 대한 자세한 설명은 **program implementation**에서 설명한다.

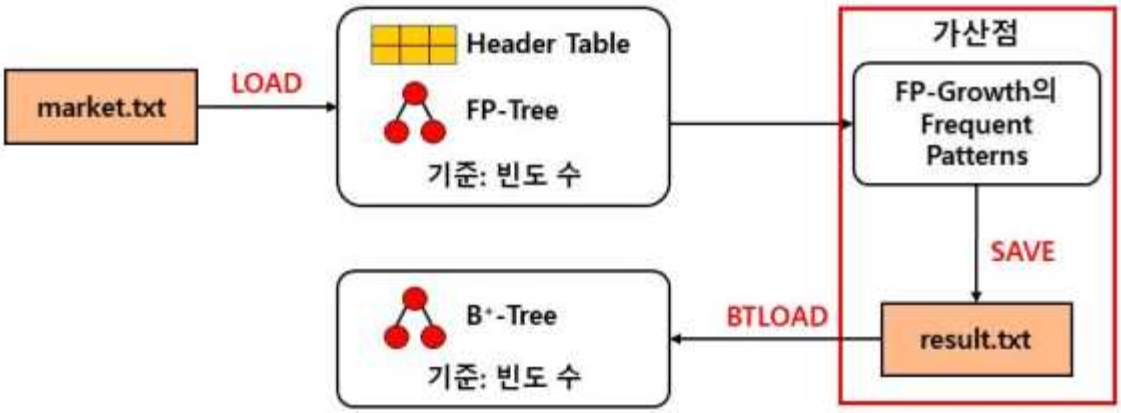

그림 1. 상품 추천 프로그램 구조

☐ Program implementation

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. 이를 구현하는 경우 가산점이 부여된다.

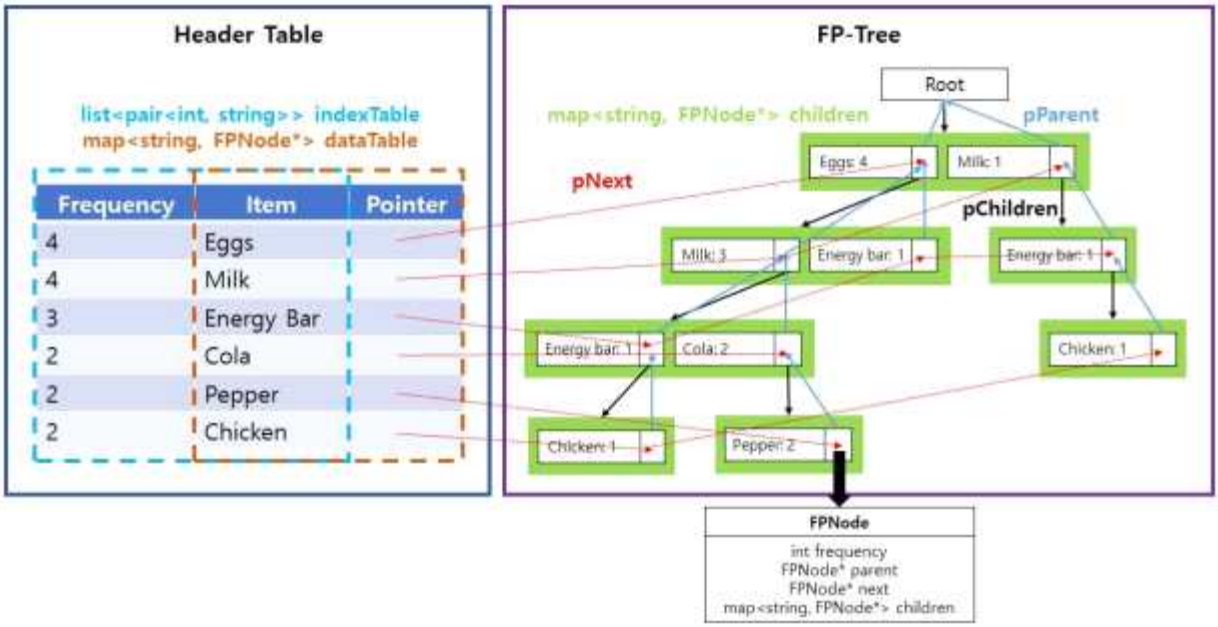

그림 2. FP-Growth의 예

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.

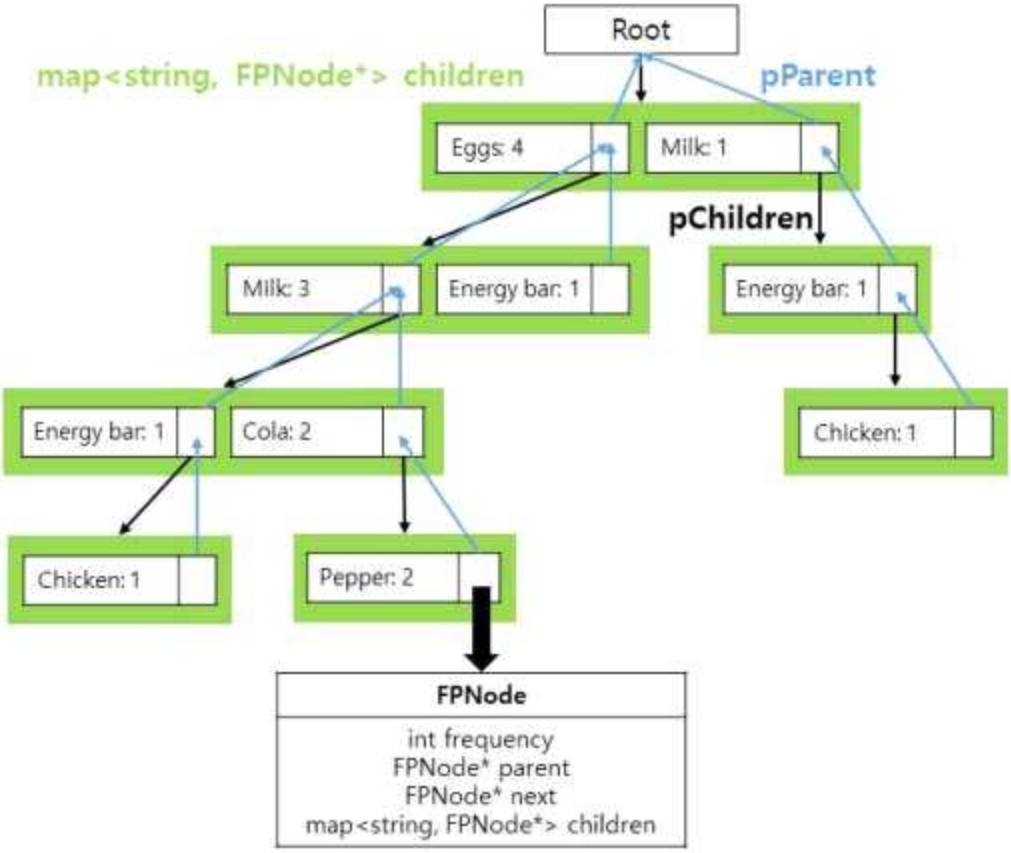

그림 3. FP-Tree의 예

- 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.

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.

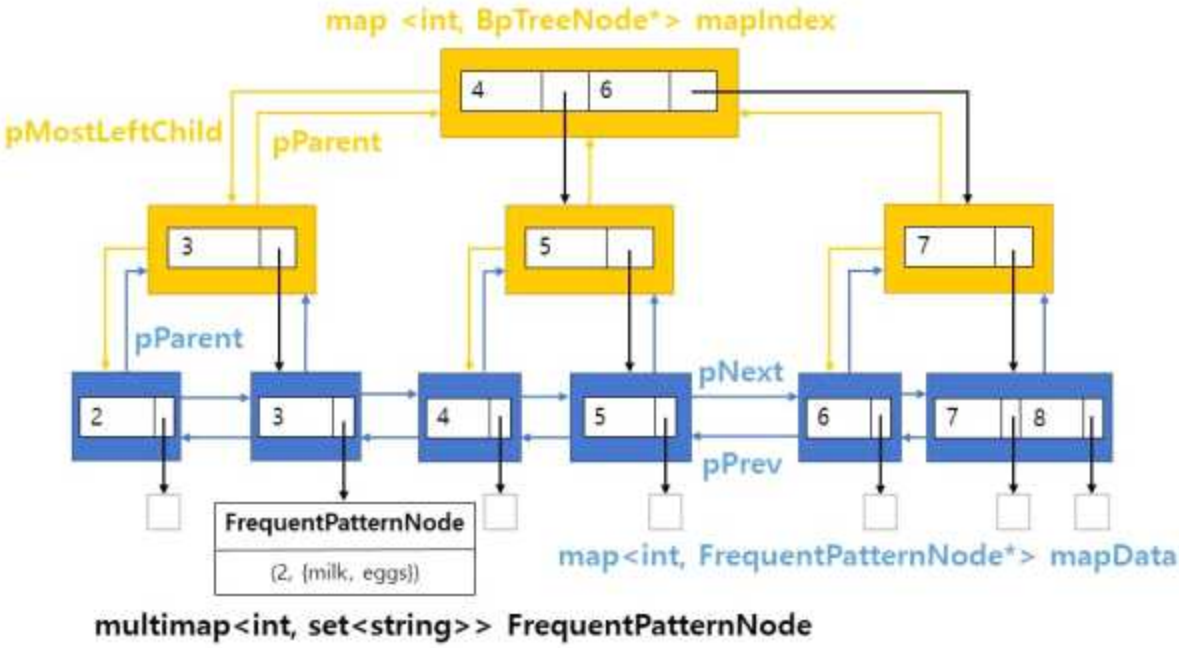

그림 4. B⁺-Tree의 예

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.

☐ Functional Requirements

- 출력 포맷은 포맷에 대한 예시일 뿐 실제 출력되는 데이터들과는 차이가 있을 수 있습니다.

표 1. 명령어 사용 예 및 기능 설명

명령어	명령어 사용 예 및 기능
LOAD	사용 예) LOAD 텍스트 파일의 데이터 정보를 불러오는 명령어로, 텍스트 파일에 데이터가 존재하는 경우 텍스트 파일을 읽어 FP-Growth를 생성한다. 만약 텍스트 파일이 존재하지 않거나 자료구조에 이미 데이터가 들어가 있으면 알맞은 에러 코드를 출력한다. 사용되는 텍스트 파일의 이름은 아래와 같으며 파일 이름을 수정하지 않는다.

	텍스트 파일: market.txt **•데이터 조건** - 상품명은 무조건 소문자로 주어지며 그 이외의 입력은 없다고 가정한다. - "\t"를 구분자로 하여 상품을 구분한다. - 같이 구매한 물품들은 줄 단위로 구분된다. 출력 포맷 예시) ========LOAD======== Success ======================== ========LOAD======== ERROR 100 ========================
BTLOAD	사용 예) BTLOAD 텍스트 파일의 데이터 정보를 불러오는 명령어로, 텍스트 파일에 데이터 정보가 존재하는 경우 텍스트 파일을 읽어 B+-Tree에 저장한다. 만약 텍스트 파일이 존재하지 않거나 자료구조에 이미 데이터가 들어가 있으면 알맞은 에러 코드를 출력한다. 사용되는 텍스트 파일의 이름은 아래와 같으며 파일 이름을 수정하지 않는다. 텍스트 파일: result.txt **•데이터 조건** - 첫 번째 값은 빈도수고 그 이후 값은 상품 정보다. - "\t"를 구분자로 하여 상품을 구분한다. - Frequent Pattern에 속한 상품들은 줄 단위로 구분된다. 출력 포맷 예시) ========BTLOAD======== Success ======================== ========BTLOAD======== ERROR 200 ========================

PRINT_ITEMLIST	사용 예) 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. 출력 포맷 예시) ========PRINT_ITEMLIST======== Item Frequency eggs 4 milk 4 energy bar 3 cola 2 pepper 2 chicken 2 ========== ========PRINT_ITEMLIST======== ERROR 300 ==========
PRINT_FPTREE	사용 예) 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. **•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. 출력 포맷 예시) ========PRINT_FPTREE======== {StandardItem,Frequency} (Path_Item,Frequency) {chicken, 2} (chicken, 1) (energy bar, 1) (milk, 3) (eggs, 4)

	(chicken, 1) (energy bar, 1) (milk, 1) {pepper, 2} (pepper, 2) (cola, 2) (milk, 3) (eggs, 4) {cola, 2} (cola, 2) (milk, 3) (eggs, 4) {energy bar, 3} (energy bar, 1) (milk, 3) (eggs, 4) (energy bar, 1) (eggs, 4) (energy bar, 1) (milk, 1) {milk, 4} (milk, 3) (eggs, 4) (milk, 1) {eggs, 4} (eggs, 4) ========== ========PRINT_FPTREE======== ERROR 400 ==========
PRINT_BPTREE	사용 예) PRINT_BPTREE eggs 2 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 에러 코드를 출력한다. 출력 포맷 예시) ========PRINT_BPTREE======== FrequentPattern Frequency {pepper, eggs} 2 {pepper, cola, eggs} 2 {pepper, milk, eggs} 2 {pepper, cola, milk, eggs} 2 {cola, eggs} 2 {cola, milk, eggs} 2

	{energy bar, eggs} 2 {milk, eggs} 3 ========== ========PRINT_BPTREE======== ERROR 500 ==========
PRINT_CONFIDENCE	사용 예) PRINT_CONFIDENCE milk 0.6 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. 연관율은 $\frac{\text{부분집합의 빈도수}}{\text{해당 상품의 총 빈도수}}$ 로 계산이 된다. 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. 출력 포맷 예시) ========PRINT_CONFIDENCE======== FrequentPattern Frequency Confidence {milk, eggs} 3 0.75 ========== ========PRINT_CONFIDENCE======== ERROR 600 ==========
PRINT_RANGE	사용 예) PRINT_RANGE milk 3 4 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 Frequent Pattern을 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

<table>
<tr><td></td><td>코드를 출력한다.

출력 포맷 예시)
========PRINT_RANGE========
FrequentPattern Frequency
{milk, eggs} 3
==========

========PRINT_RANGE========
ERROR 700
==========</td></tr>
<tr><td>SAVE</td><td>사용 예) 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.

출력 포맷 예시)
=======SAVE========
Success
==========

========SAVE========
ERROR 800
==========

result.txt 결과 예시)
2 chicken energy bar
2 chicken milk
2 chicken energy bar milk
2 pepper cola
2 pepper milk
2 pepper eggs
2 pepper cola milk
2 pepper cola eggs
2 pepper milk eggs
2 pepper cola milk eggs
2 cola milk
2 cola eggs
2 cola milk eggs
2 energy bar milk
2 energy bar eggs
3 milk eggs</td></tr>
</table>

EXIT	사용 예) EXIT 프로그램상의 메모리를 해제하며, 프로그램을 종료한다. 출력 포맷 예시) =======EXIT======== Success ==========

☐ Requirements in implementation

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

☐ 동작 별 에러 코드

동작	에러 코드
LOAD	100
BTLOAD	200
PRINT_ITEMLIST	300
PRINT_FPTREE	400
PRINT_BPTREE	500
PRINT_CONFIDENCE	600
PRINT_RANGE	700
SAVE	800

□ 구현 시 반드시 정의해야하는 Class 및 멤버 변수

1. FPGrowth : FP-Growth 클래스

항목	내용	비고
threshold	빈도수 제한	멤버 변수로 변경 가능, 2 이상의 값을 가짐
fpTree	FP-Tree	자식 노드를 제외하고 NULL 값을 가짐
table	Header Table	
frequentPattern	생성된 Frequent Patterns	공집합과 집합의 크기가 1인 Frequent Pattern은 제외
fout	log 파일	
flog	result 파일	소수점 2자리까지만 입력 가능

2. FPNode: 아이템 정보 및 FP-Growth 노드

항목	내용	비고
frequency	상품 빈도수	
parent	부모 노드	
next	다음 FPNode를 가리킴	
children	자식 노드를 가지고 있는 map 컨테이너	map<string, FPNode*> (상품명, 하위 상품 노드*)

3. HeaderTable: FP-Growth의 Header Table 클래스

항목	내용	비고
indexTable	상품 빈도수 및 상품명을 pair로 저장하고 있는 list	list<pair<int, string>> (상품의 빈도수, 상품 이름)
dataTable	상품 정보를 가지고 있는 map 컨테이너	map<string, FPNode*> (상품명, 하위 상품 노드*)

4. FrequentPatternNode : Frequent Pattern 노드 클래스

항목	내용	비고
frequency	Frequent Pattern의 크기	
FrequentPatternList	Frequent Pattern들을 가지고 있는 multimap 컨테이너	multimap<int, set<string>> (Frequent Pattern 크기, Frequent Pattern)

5. BpTreeNode : B$^+$-tree의 노드 클래스

항목	내용	비고
pParent	상위 BpTreeNode를 가리킴	
pMostLeftChild	하위 BpTreeNode 중 가장 왼쪽 노드를 가리킴	

6. BpTreeIndexNode : B$^+$-tree의 인덱스 노드 클래스

항목	내용	비고
mapIndex	인덱스 값을 가지고 있는 map 컨테이너	map<int, BpTreeNode*> (빈도수, 하위 B$^+$-Tree 노드*)

7. BpTreeDataNode : B+-tree의 데이터 노드 클래스

항목	내용	비고
mapData	해당 빈도수에 속한 Frequent Pattern들을 가지고 있는 map 컨테이너	map<int, FrequentPatternNode*> (빈도수, Frequent Pattern 정보 객체*)
pNext	다음 BpTreeNode를 가리킴	
pPrev	이전 BpTreeNode를 가리킴	

8. BpTree : B+-tree 클래스

항목	내용	비고
root	B+-tree의 루트	
order	B+-tree의 차수(m)	멤버 변수로 변경 가능

9. Manager : 다른 클래스들의 동작을 관리하여 프로그램을 전체적으로 조정하는 역할을 수행

항목	내용	비고
fpgrowth	FP-Growth	첫 번째 인자(threshold)를 받음
bptree	B+-tree	두 번째 인자(order)를 받음

□ Files

- ✓ market.txt : 프로그램에 추가할 장바구니 데이터가 저장된 파일
- ✓ result.txt : 연관 상품 집합들을 저장하고 있는 파일
- ✓ command.txt : 프로그램을 동작시키는 명령어들을 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 구현하며 작성된 소스 파일의 이름과 클래스와 함수 이름 및 형태를 임의로 변경하지 않는다.
- ✓ 클래스의 함수 및 변수는 자유롭게 추가 구현이 가능하다.
- ✓ 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.
- ✓ 채점은 제공된 파일이 아닌 다른 1000개 이상의 구매 상품 목록을 사용하여 진행된다.
- ✓ FP-Growth는 FP-Tree와 HeaderTable 연결까지가 채점 기준이고 result.txt는 FP-Growth의 결과물인 Frequent Pattern들로 result.txt 생성까지 구현하는 경우 가산점이 제공된다. 구현을 못 한 학생들을 위해 따로 result.txt를 제공한다.
- ✓ 제공되는 testcase 파일과 result 파일은 각각 market.txt와 result.txt로 변경해서 사용한다.
  ※ (testcase1, result1), (testcase2, result2), (testcase3, result3)

- 추가 공지사항

Notice

Notice !! 윈도우에서 리눅스로 단순 파일 복사 및 드래그는 파일 인코딩 변환 문제가 발생할 수 있으니 반드시 리눅스 환경에서 파일을 재생성 후 테스트 바랍니다.

Notice !!설계와 실습을 모두 수강하시는 분은 실습 과제제출란에만 제출바랍니다.

Notice !!log.txt의 출력 포맷에서 모든 구분자는 '\t'이 아닌 스페이스바(' ')로 하시면 됩니다.

Notice !! 채점 진행시 command.txt의 인자 구분자는 '\t'을 사용하여 진행하도록 하겠습니다.

Knowledge

아래 명령어 예시에서 앞의 $ 로 시작되는 부분은 명령어 입력 부분이고, 그 외 는 출력 부분임

리눅스 명령어 요약

1. ls : list로 현재 작업중인 디렉토리의 파일 및 포함된 디렉토리 목록들을 표시 (-a, -l 속성으로 자세한 출력 가능)
2. pwd : print working directory로 현재 작업중인 디렉토리의 절대경로 위치 출력
3. cd : change directory로 디렉토리 를 변경(. : 현재 디렉토리, .. : 상위 디렉토리)

```
$ ls
Documents Download
$ ls -l
drwxr-xr-x 2 user user     4096 Oct 05  2020 Documents
drwxr-xr-x 2 user user     4096 Oct 05  2020 Downloads
$ pwd
/home/user
$ cd Download
$ pwd
/home/user/Downloads
```

requirement

먼저 해당 github에 저장되어 있는 base 코드를 다운받는다.

```
$ sudo apt-get install git
$ git clone https://github.com/DSLDataStorage/DS_Project_2_2022_2.git
```

how to compile this project

make명령어 실행 후 ls명령어를 통해 해당 디렉토리를 확인해 보면 run 이라는 파일이 생긴것을 확인 할 수 있다.

$ make
g++ -std=c++11 -g -o run HeaderTable.cpp FPNode.cpp BpTree.cpp Manager.cpp main.cpp FPGrowth.cpp BpTree.h Manager.h FrequentPatternNode.h BpTreeNode.h FPGrowth.h BpTreeIndexNode.h HeaderTable.h FPNode.h BpTreeDataNode.h

$ ls
BpTree.cpp BpTreeNode.h FPNode.cpp main.cpp market.txt BpTreeDataNode.h command.txt FPNode.h makefile BpTree.h FPGrowth.cpp HeaderTable.cpp Manager.cpp FrequentPatternNode.h BpTreeIndexNode.h FPGrowth.h HeaderTable.h Manager.h **run**

how to run code

./(생성된 실행파일) 의 형식으로 생성된 run 실행파일을 실행한다.

실행하면 결과로 result.txt파일과 log.txt파일이 생성되면서 결과가 result.txt와 log.txt에 저장된다.

```
$ ./run
$ cat log.txt
==> command 1) LOAD
Success
```