데이터구조 및 설계 2차 프로젝트 보고서

2019202050 이강현

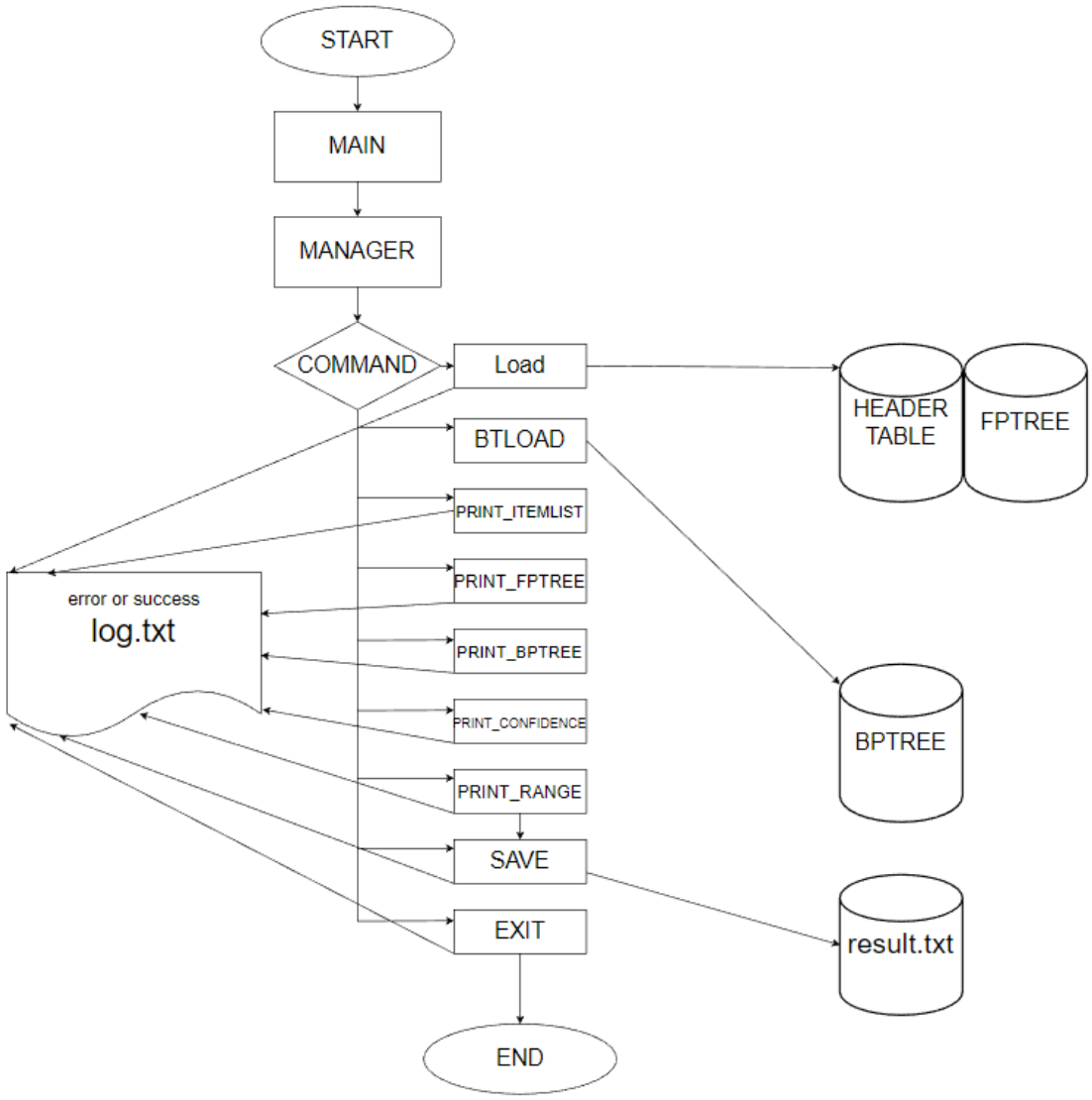
1. Introduction

이번 프로젝트는 FP-Growth와 B+ Tree를 이용하여 상품 추천프로그램을 구현하는 것이다. Market.txt에서 개개인이 구매한 상품들의 목록을 불러와 FP-Growth를 구축한 후 연관성을 확인한다. FP-Growth는 상품의 빈도수와 이름을 저장하는 index table, 상품의 이름과 FP-Tree의 노드를 연결한 포인터가 저장되어있는 data table을 합친 header table과 상품들의 연관성을 저장하고 있는 FP-Tree로 구성된다. FP-Growth를 통해 연관된 상품들로 구성된 frequent pattern들을 result.txt에 저장하고 이를 근거로 B+ Tree를 구성한다.

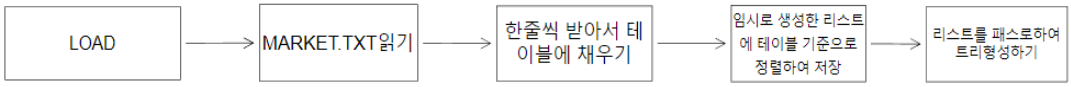
B+ Tree는 indexnode와 datanode로 구성이 되고 indextable은 datanode를 탐색하기 위해 사용되고 datanode는 빈도수와 frequent pattern들이 저장된 frequentpatternnode를 가리키는 포인터로 구성되어 frequentpattern들이 저장되어있는 곳으로 사용된다.

이러한 자료구조를 구축하기 위해 node들간의 연결상태를 나타낼 수 있는 linked list구조에 대한 이해, map,multimap,set,vector등 stl로 제공되는 구조들의 이해와 사용법등이 필요하다. 또한 SAVE명령어를 통해 상품들간의 관련성을 저장하는데 필요한 부분집합과 멱집합 같은 집합에 대한 수학적 이해가 필요하다. 또한 저장된 파일들을 읽고 쓰는 작업을 하기위해 스트림에 대한 이해와 부분집합의 수와 상품의 빈도수를 통해 구할 수 있는 연관율에 대한 이해등 기본적인 개념들이 필요할 것으로 보인다. 프로젝트는 LOAD, BTLOAD, PRINT_ITEMLIST, PRINT_FPTREE, PRINT_BPTREE, PRINT_CONFIDENCE, PRINT_RANGE, SAVE, EXIT로 구성되며 각각의 명령어는 txt파일에서 불러와 명령을 실행한 뒤 log.txt에 저장하여 프로그램의 진행상태를 알 수 있게한다.

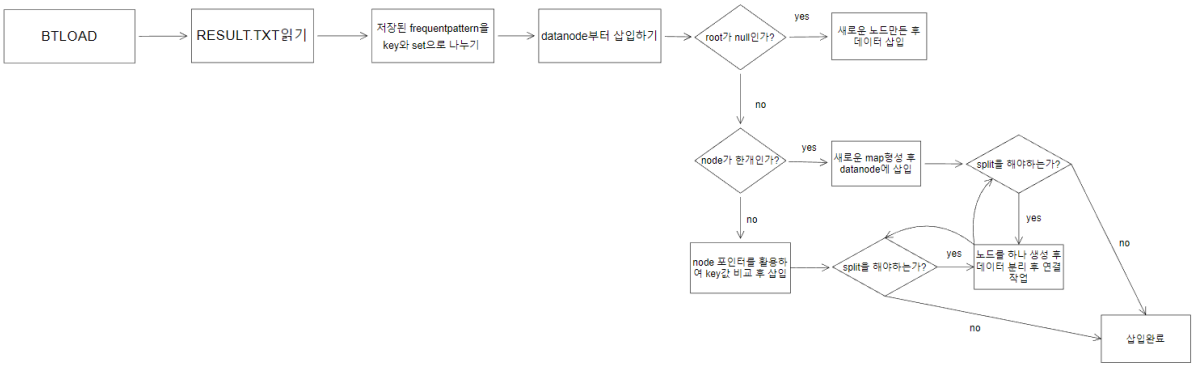
2. Flowchart



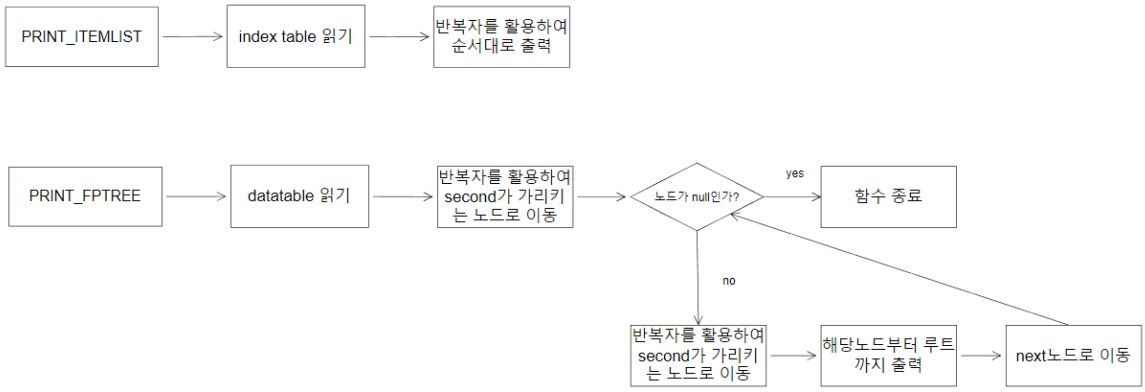
전체적인 프로그램의 흐름은 위와 같다. LOAD명령어를 통해 FPTREE를 구축하고 BTLOAD 명령어를 통해 BPTREE를 구축한다. 각각의 출력명령어를 통해 TREE와 TABLE의 내용들을 log.txt에 저장하고 SAVE명령어를 통해서는 FP-Growth내에서 구한 frequent pattern들을 result.txt에 저장한다.



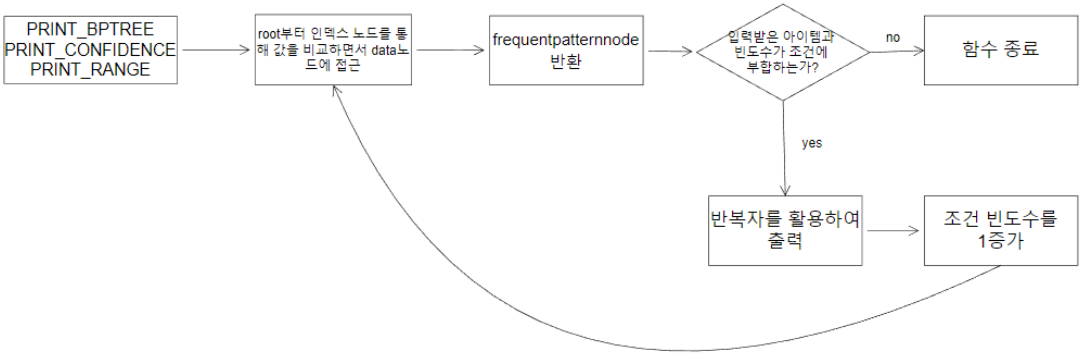
먼저 LOAD명령어이다. LOAD명령어는 market.txt에 저장된 구매목록을 한줄씩 읽어오고 strtok을 통해 잘라낸 후 빈도수와 상품명으로 먼저 indextable에 저장하게 된다. 구현한 프로그램에서는 indextable을 구성한 후 한번 더 market.txt을 읽어 list형태로 구매내역을 한줄 단위로 구분하여 저장한 뒤 이를 미리 구성한 indextable의 빈도수를 기준으로 정렬한 뒤 FP-Tree를 구성하는데 사용하였다. 이때 list는 구매내역임과 동시에 FP-tree의 루트부터 단말노드까지의 하나의 경로가 된다.



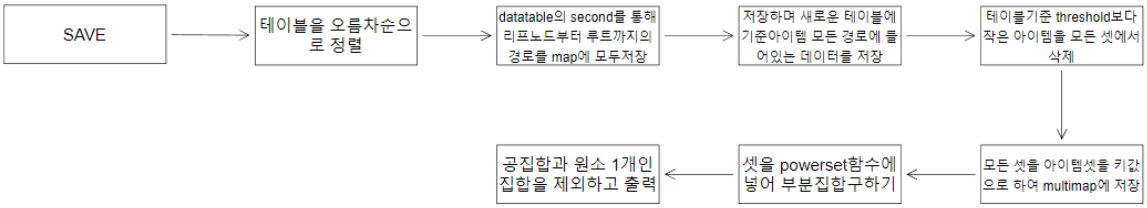
BTLOAD명령어이다. BTLOAD명령어는 result.txt에 저장된 frequentpattern들을 b+tree에 저장하는 명령어이다. Result.txt를 먼저 읽은 후 frequentpattern을 key와 set으로 나누어 root부터 mostleftchild를 통해 가장 아래 datanode에 접근한 후 데이터를 삽입한다. 이렇게하는 근거는 b+tree의 order때문이다. Order의 크기에 따라 노드의 크기가 결정되는데 크기를 넘게 되면 split을 해야한다. Split을 하게되면 indexnode가 추가되기에 실질적인 datanode를 통해 indexnode를 구현하게 되었다. 분기는 root가 null일 때, node가 한개일때 따라서 indexnode가 없을 때, 노드도 여러 개고 indexnode또한 1개 이상일때로 나누어 코드를 구현하였다.



FP-Growth를 출력하는 명령어는 위의 두가지이다. PRINT_ITEMLIST는 indextable을 순차적으로 출력하는 것이고, PRINT_FPTREE는 datatable에 저장된 상품마다 가지는 노드포인터를 통해 연결된 노드로부터 루트까지 순차적으로 상품들을 출력하는 명령어이다. 코드 내에서는 루트는 일반 노드들과 같은 노드로 이루어져있고 노드의 데이터가 없는 상태이므로 frequency가 0인 노드를 만날때까지를 반복문 탈출 조건으로 하여 구현하였다.



위의 명령어들은 B+ Tree내의 데이터를 출력하는 명령어이다. 이는 인덱스노드들은 map<int,BpTreeNode*>형태로 데이터가 저장되어있으므로 반복자를 사용하여 first, 즉 int값을 비교하면서 datanode에 접근하고 datanode까지 왔다는 것을 알기위해서는 datanode와 indexnode의 차이점 mostleftchild이 있는지를 확인하여 반복문을 탈출한다. 그 후 datanode에 접근하면 datanode내의 map<int, frequentpatternnode*>에서 또 반복자를 사용하여 값을 비교하고 입력받은 값과 조건이 일치하면 frequentpatternnode를 반환하는 함수를 따로 구현하였고 이를 통해 값들을 출력하였다.



마지막으로 SAVE이다. SAVE는 테이블을 오름차순으로 정렬한뒤 PRINT_FPTREE를 출력했던 방식과 동일하게 접근하여 경로들을 map에 모두 저장한다. 경로를 저장할때는 datatable가 가지고 있는 상품중 하나를 기준으로 잡고 next포인터로 연결된 해당 상품들을 출발점으로하여 루트까지의 경로를 하나의 셋으로 하여 여러 개의 셋을 저장한다.

셋내의 모든상품들은 기준이되는 상품의 빈도수로 통일되어 저장된다. 그 이후 하나의 상품을 기준으로 하여 만들어진 새로운 테이블을 기준으로 threshold보다 작은 상품들은 모든 셋 내에서 삭제한다. 그렇게하여 남겨진셋들은 중복이 될 수 있으므로 multimap에 저장한다. 그 후 powerset함수를 통해 부분집합을 구하고 이번에는 map에 공집합과 원소 1개인 집합을 제외하고 result.txt에 기록한다.

3. Algorithm

이번 프로젝트내에서 사용된 알고리즘은 FP-Growth와 B+ Tree로 크게 두가지 자료구조가 사용되었다. 이는 구매되는 상품들의 관련성에 깊게 연관되어 있고 빅데이터를 처리하는데 기본이된다. 또한 STL을 많이 사용하기 때문에 각각의 STL에 데이터가 어떻게 저장되는지를 알아야 한다. 이번 프로젝트에서는 각각의 트리구조에서 삽입만을 다루기 때문에 어떻게 삽입을 하고 접근하는지 위주로 설명하겠다.

우선 FP-Growth의 Header Table이다. Header table은 list<pair<int,string>>구조인 indextable과 map<string,FpNode*>구조인 datatable로 구성되어있다. 이는 데이터의 빈도수와 데이터정보를 pair로 묶어 저장하기 때문에 정렬된 형태로 데이터를 제공받을 수 있고 양방향리스트로 저장된 컨테이너형태이므로 리스트의 앞과 뒤에 데이터를 삽입하고 삭제할 수 있다. 또한 반복자를 통해 순차적으로 접근하기 때문에 모든 데이터를 사용할 때 활용도가 높다. Datatable의 map은 balanced binary search tree로 key와 value로 저장이되는 컨테이너이다. Map은 key의 중복을 허용하지 않아 key가 독립적인 위치를 가지게 된다는 점이 높은 활용도를 보인다. 또한 list와는 다르게 key값을 통해 바로 value값을 탐색할 수 있다는 점이 사용할 때 편리함을 제공한다.

FP-Tree는 datatable의 value값인 노드포인터와 링크드리스트형태로 연결된 트리구조이며 따라서 루트에서 접근할 때는 구매목록을 알 수 있고 테이블에서 접근할 때는 모든

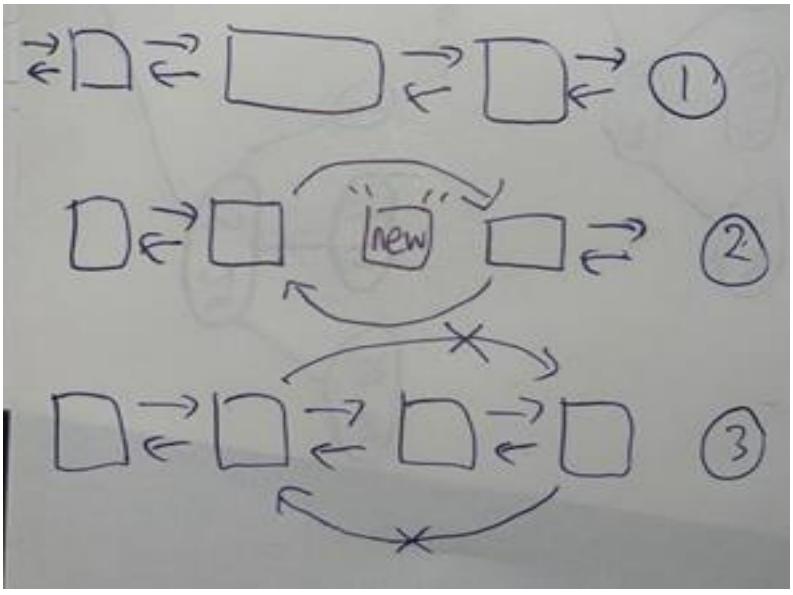
노드의 수와 그 노드를 기준으로 하여 관련성정보를 얻을 수 있는 자료구조임을 알 수 있다.

B+ Tree는 조건에 맞는 데이터를 탐색하기 위한 노드 indexnode와 실제 데이터셋들을 가리키고 있는 datanode로 크게 구성되어있고 datanode의 map의 value값은 셋들을 저장하고 있는 frequentpatternnode를 가리키는 형태로 구성되어 있다. 또한 frequentpatternode는 multimap형태로 데이터셋들을 저장하며 이는 같은 빈도수를 가진 셋들이 많이 저장되어야 하기 때문에 key값의 중복이 가능한 multimap을 사용하였다. datanode들은 서로 모두 양방향리스트로 연결되어있기 때문에 모든 데이터 접근이 용이하고 특정값을 가지는 노드를 찾기 위해서는 루트부터 indextnode를 사용하여 접근하면 더 빠르게 값을 찾을 수 있는 구조이다.

이번에는 프로젝트를 구현하면서 개인적으로 어려웠고 중요하다고 생각되는 B+ Tree의 split작업과 FP-Growth의 SAVE작업을 설명하고자 한다.

Split은 B+Tree에서 노드가 가질 수 있는 데이터의 한계치를 넘을 때 새로운 노드와 값을 나눠 가지며 부모를 형성하여 부모가 노드들을 가리킬 수 있게끔하는 작업이다.

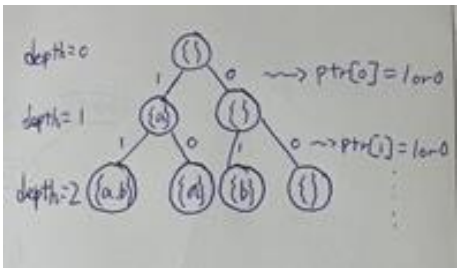
Split을 할 데이터의 위치는 수업시간때 제공받은 pdf를 기준으로 분할하였고 새로운 노드를 생성하고 데이터의 뒤쪽부터 접근하여 데이터를 하나씩 기존노드에서 삭제하고 새로운노드에 삽입하였다. 데이터의 분할이 끝나면 기존노드의 parent가 있다면 새로운 노드또한 동일한 parent를 가지므로 가리키게끔하고 분할하며 생성된 새로운 indexnode의 데이터는 새로운 노드를 가리키게끔 구성하였다. Split을 구현하며 한가지 중요하다고 느낀 작업이 있었는데 새로운 노드의 next와 prev를 잘 설정하는 것이다.

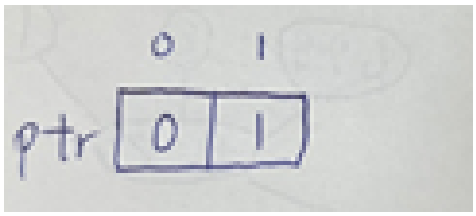


위와같이 연결해주지 않고 단순히 분할한 노드끼리 서로 분할하게 되면 트리가 많이 꼬이게 되므로 주의해야 한다.

SAVE를 구현할 때는 powerset함수를 사용하기 전에 데이터를 전처리하는 것이 중요했는데 데이터를 트리에서 뽑아낸 후 이를 테이블을 기준으로 threshold 아래인 아이템은 제거하고 이를 multimap에 저장하고 저장할 때에는 셋을 key값으로 하여 저장하여 동일한 키값을 받을 수 있게끔 하고 다시 map에 저장하여 없는 셋이라면 추가, 있는 셋이라면 빈도수를 업데이트시켜주며 그렇게 정제된 데이터를 powerset에 넣고 공집합과 원소 1개인 집합을 제거하였다. Multimap과 map, vector등의 stl을 적절하게 사용하여야 했고 또한 powerset함수는 data를 저장할 map, 하나의 dataset, 기준이 되는 item, 셋의 빈도수, 데이터의 개수만큼의 크기를 가지는 배열, 깊이를 인자로 받는다.

먼저 함수가 시작되면 set<string>을 부분집합을 담아둘 통으로 생각하고 기준이되는 item을 먼저 삽입한채로 시작한다. 그 이후 깊이가 데이터의 크기와 같아지면 set을 배열내의 정보를 기준으로 부분집합을 만들고 map에 저장한다. 재귀적으로 구현된 함수이기때문에 함수가 시작되면 같은함수가 데이터의 크기만큼 두번씩 실행된다.





위는 예시이다 위와 같은 상태트리를 가지고 깊이가 데이터의 크기 즉 트리 맨 아래에 도달하면 ptr배열에 저장된 정보를 토대로 부분집합을 구성한다.

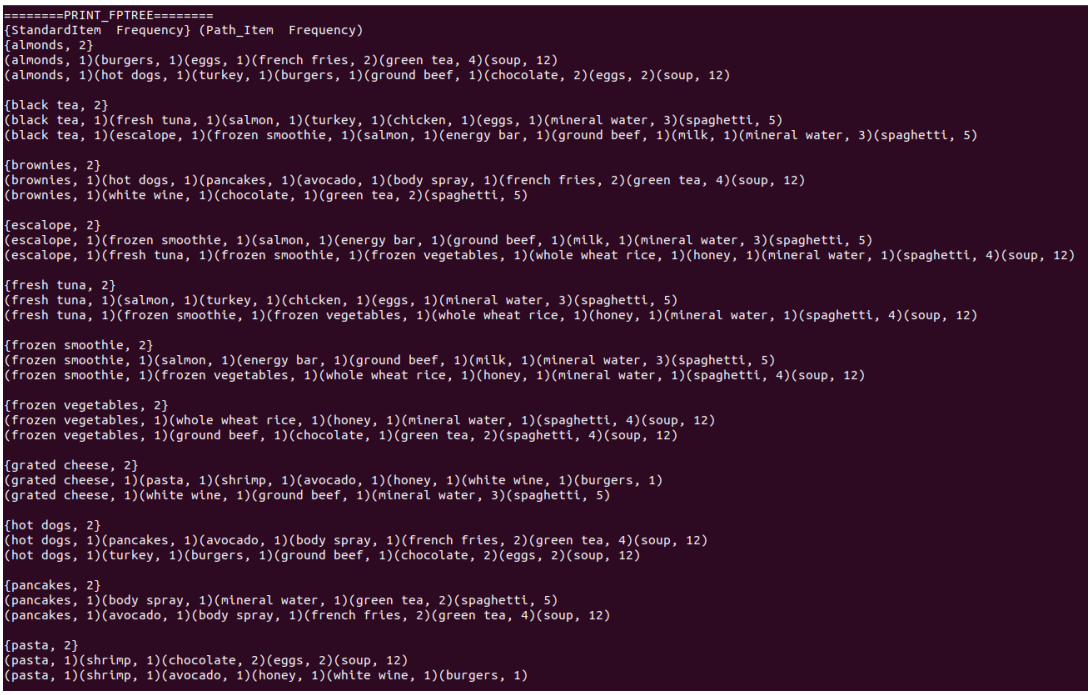
4. Result screen

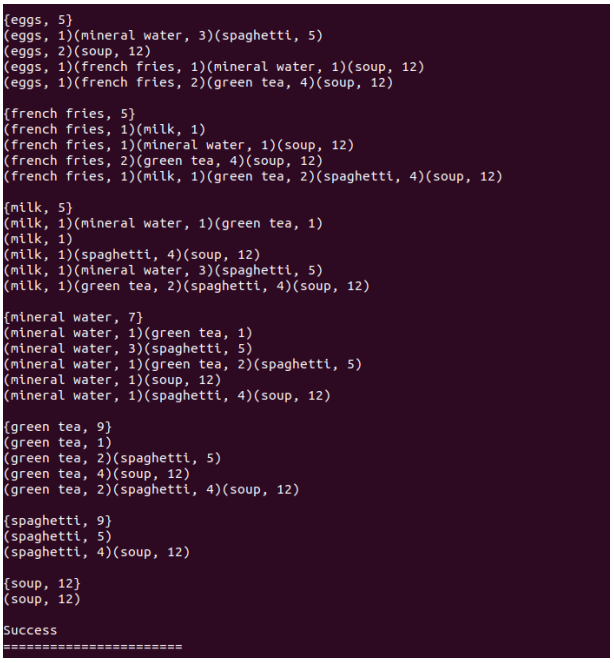
결과는 아래와 같은 명령어 순서로 보여진다.

```
PRINT_FPTREE
PRINT_ITEMLIST
SAVE
PRINT_BPTREE    eggs    2
PRINT_CONFIDENCE        milk    0.2
PRINT_RANGE     french fries    3       4
LOAD
PRINT_FPTREE
PRINT_ITEMLIST
SAVE
BTLOAD
PRINT_BPTREE    eggs    2
PRINT_CONFIDENCE        milk    0.6
PRINT_RANGE     milk    3       4
EXITlee@ubuntu:~/Desktop/data_structure_design2$
```

우선 에러가 나는 부분과 LOAD가 잘 실행된 모습을 볼 수 있다.

Print와 SAVE명령어는 자료구조에 데이터가 없는 상태이므로 에러코드를 출력하는 것을 볼 수 있다.





LOAD가 정상적으로 실행된 이후 fptree를 출력하는 모습이다

오름차순으로 정렬되어 트리의 데이터를 잘 출력하였다.

Itemlist의 출력모습이다.

```
========SAVE========
Success
=======================

========BTLOAD========
Success
=======================

========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
Success
=======================

========PRINT_CONFIDENCE========
FrequentPattern  Frequency  Confidence
{  milk, spaghetti }   3   0.6
Success
=======================

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

========EXIT========
success
====================
```

SAVE를 마무리하고 명령어를 통해 생성된 result.txt에서 데이터를 불러와 BTLOAD또한 실행이 잘 되는 것을 확인할 수 있다. Egg가 들어있는 셋들중 빈도수가 2이상인 셋들이 명령에 맞게 잘 보여진다.

Confidence또한 milk가 포함되고 0.6의 연관율 이상을 가지는 셋이 출력되었다.

Range도 3과 4의 빈도수 내에 milk가 포함된 셋이 출력되었다.

결과는 testcase1과 직접 구현한 result.txt를 이용하여 검증을 완료했다.

5. Consideration

이번 프로젝트에서는 여러가지를 구현하는 작업보다는 자료구조를 구현하고 구조가 잘 만들어졌는지 확인하는 출력작업이 더 많았던 것 같다. FP-Growth와 B+ Tree를 구현하는 것은 이전의 코드작업처럼 링크드리스트를 구현하거나 힙을 구현하거나 하는 한가지만 구현하는 것이 아니라 필요할 때 사용이 가능해야 했다.

또한 STL을 활용할 줄 알아야 했는데 map,set,multimap,vector,list등 다양한 STL의 개념을 학습함과 동시에 실제로 어떻게 사용되는지를 배울 수 있었다.

LOAD와 BTLOAD가 자료구조를 구축하는 명령어였는데 테이블을 구축하는 것은 비교적 쉬웠으나 fptree를 구현할 때 구매목록을 테이블 빈도수를 기준으로 정렬하여 삽입해야했는데 이를 어떻게 해야할지 고민을 했던 것 같다. 결국 입력스트림을 닫고 다시 열어서 한줄씩 받아오고 list를 사용하여 정렬한 뒤 저장하는 방법을 사용하였다.

B+ Tree는 알고리즘을 설명할 때 다룬 split부분에서 split을 하고난 뒤 next와 prev를 설정하는 것을 하지 않아 데이터가 손실되는 경험을 하였고 트리를 구축할때도 부모노드와의 연결, 분할한 후 기존노드에 연결된 노드들을 새로운 노드에 연결하는 작업들을 order를 얼마나 입력받든 분할을 많이 하든 언제든지 적용이 가능하게 끔 함수를 일반화하려는 것이 많이 힘들었던 것 같다.

Print하는 함수들은 트리를 구축할 때 사용한 함수들을 이용하여 쉽게 함수를 제작할 수 있었던 것 같다. 다만 b+ Tree를 출력할 때는 데이터를 삽입할때와는 다르게 인덱스노드를 활용해야 했기 때문에 데이터를 비교하면서 내려가는 것이 stl사용이 미숙해서 그런지 어려웠다.

또한 출력하는 부분에서 confidence의 값이 잘 나오지 않는 경우가 발생하였는데 이는 출력하는 데이터셋을 가져오는 함수를 구현했으나 입력받은 값을 찾지 못한다면 null값을 반환하게끔하여 데이터가 출력되다가 데이터가 없는 빈도수를 마주하면 그대로 종료되는 경우가 생겼다. 따라서 minimum과 maximum함수를 통해 데이터가 저장된 트리내에 빈도수의 최대값과 최솟값을 반환해주어 그 사이에 있다면 continue문을 통해 반복문을 계속 돌게 끔 구현하였다. 또한 confidence가 매우 작은 값이 들어오게 되면 최솟값과 최대값 범위내에도 있지 않기 때문에 초기 출력 시작값은 입력인자가 최솟값보다 작다면 최솟값부터 출력을 시작하게끔 코드를 구현하였다.

마지막으로 SAVE명령어가 testcase1에서는 result1과 동일한 결과를 냄을 알 수 있었으나 testcase2와 testcase3에서는 오차가 전자는 15정도 후자는 6정도의 셋의 개수가 추가적으로 저장됨을 확인할 수 있었다. 이를 excel에서 직접 값을 비교해 본 결과 모든 셋을 눈으로 확인하진 못했으나 아래와 같이 저장되어 있음을 확인할 수 있었다.

3362	14 burgers	cookies	eggs				14 burgers	cookies	eggs			
3363	8 burgers	cookies	eggs	french fries			8 burgers	cookies	eggs	french fries		
3364	5 burgers	cookies	eggs	french frie:	green tea		5 burgers	cookies	eggs	french frie:	green tea	
3365	2 burgers	cookies	eggs	french frie:	green tea	ham	2 burgers	cookies	eggs	french frie	green tea	milk
3366	2 burgers	cookies	eggs	french frie:	green tea	ham	2 burgers	cookies	eggs	french frie:	ham	
3367	2 burgers	cookies	eggs	french frie:	green tea	milk	3 burgers	cookies	eggs	french frie:	milk	
3368	3 burgers	cookies	eggs	french frie:	ham		2 burgers	cookies	eggs	french frie:	milk	spaghetti
3369	2 burgers	cookies	eggs	french frie:	ham	whole whe	2 burgers	cookies	eggs	french frie:	spaghetti	
3370	3 burgers	cookies	eggs	french frie:	milk		2 burgers	cookies	eggs	frozen smoothie		
3371	2 burgers	cookies	eggs	french frie:	milk	spaghetti	9 burgers	cookies	eggs	green tea		

왼쪽이 직접 구현한 SAVE를 통해 나온 result.txt이고 오른쪽이 result2파일이다.

Testcase2를 이용하여 SAVE를 진행했고 ham이라는 단어가 중복되어 들어가 있는 것을 확인할 수 있다. SAVE명령어에서 출력을 진행할 때 map에서 공집합과 1개의 원소를 제외하고 빈도수 이상을 출력하게끔 코드를 구현했으나 위와 같은 결과는 데이터셋이 키값이기 때문에 중복될 수 없으나 중복되어 출력된것으로 보아 ham이 서로 다른 것으로 인식하는 것 같다라는 생각을 하였다. 따라서 fptree에 저장된 형태를 보기 위해 print함수를 통해 찾아본 결과 ham이 중복되어 저장되어있는 경로를 확인하였다. 경로는 아래와 같다.

```
(ham, 1)(burgers, 14)(chocolate, 205)
(ham, 1)(burgers, 9)(milk, 101)
(ham, 1)(chicken, 1)(turkey, 1)(whole whe
(ham, 1)(honey, 2)(eggs, 544)
(ham, 1)(ham, 1)(honey, 2)(eggs, 544)
(ham, 1)(chicken, 1)(whole wheat rice, 3)
```

Fptree를 잘못 설계한 것이 아닌가라는 생각이 들었지만 대부분의 케이스들은 정상적으로 출력되었고 testcase3의 결과는 30000개정도 되나 오차가 6개정도밖에 나지 않았다. 특정문자열이 다르게 인식되거나 생각하지 못한 코드내의 오류가 있을 것으로 예상된다.

프로그램을 진행하면서 자료구조를 이해하는데 시간을 반이상 사용한 것 같다. 하지만 이해하고 나니 코드를 구현하는게 좀 더 쉬워졌고 STL을 조금 더 능숙하게 사용할 수 있게 되었다.