데이터구조설계/실습 보고서
Project #2

제출 일자: 2022년 11월 20일 (일)

학과: 컴퓨터정보공학부
담당교수: 이기훈 교수님
학번: 2019202021
성 명: 정 성 엽

1. Introduction

A. 주제

FP-Growth와 B+ tree를 이용한 상품 추천 프로그램

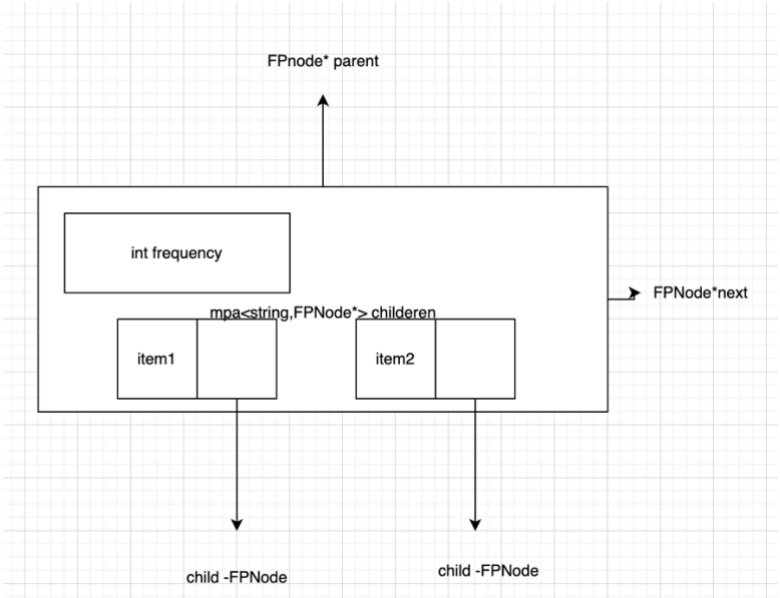
B. 목적

FP-Growth와 B+ tree를 이용하여 market.txt에 저장되어 있는 구매한 상품의 관계를 파악하여 상품 추천 프로그램을 구현한다.

C. 데이터 구조 개념

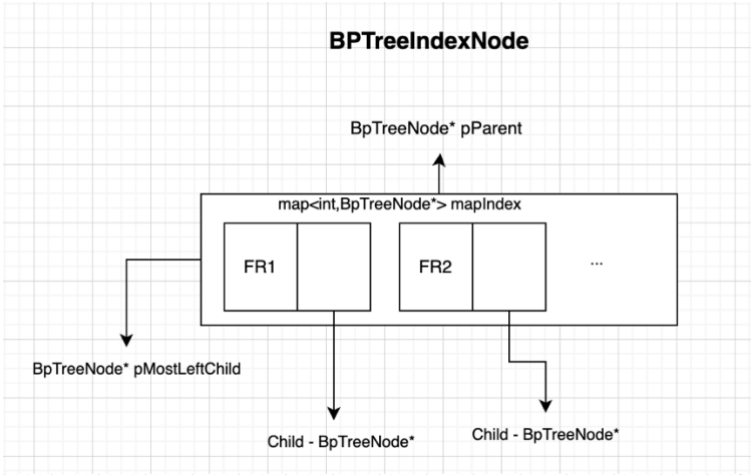
1) FP-growth

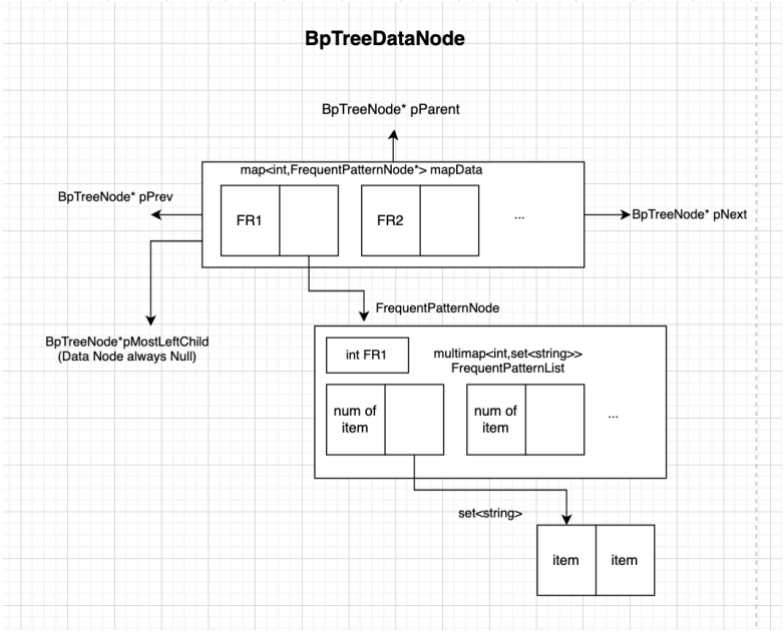
- HeadTable
  - Market.txt에 저장된 각 물품들을 이름과 빈도수를 index table에 fp tree의 각 포인터를 가리키는 pnext를 data table에 저장하고 index table과 data table은 stl을 이용하여 구현한다.
- Fp tree
  - 위의 HeadTable의 빈도수를 기준으로 market.txt를 줄 하나하나 읽어 같이 구매한 물품들을 한번에 가져오고 fp tree를 구현한다. 이때 빈도수는 입력 threshhold값 이상의 빈도수를 가진 물품만 노드에 삽입할 수 있도록 한다.



2) B+ tree

- B+ tree
  - Result.txt에 있는 빈도수와 Frequent pattern을 B+ tree에 저장하고 정렬은 빈도수를 기준으로 정렬한다. 다음 order은 변경이 가능하다.
  - BpTreeNode에 주어진 skeleton code에서는 virtual function를 통해 틀을 잡아 놓고 이후 상속을 받는다. 이때 BpIndexNode와 BpDataNode로 Node를 2개로 구분하여 구현하는데 이때의 Node들의 멤버 함수들을 그림으로 표현하면 아래와 같다.



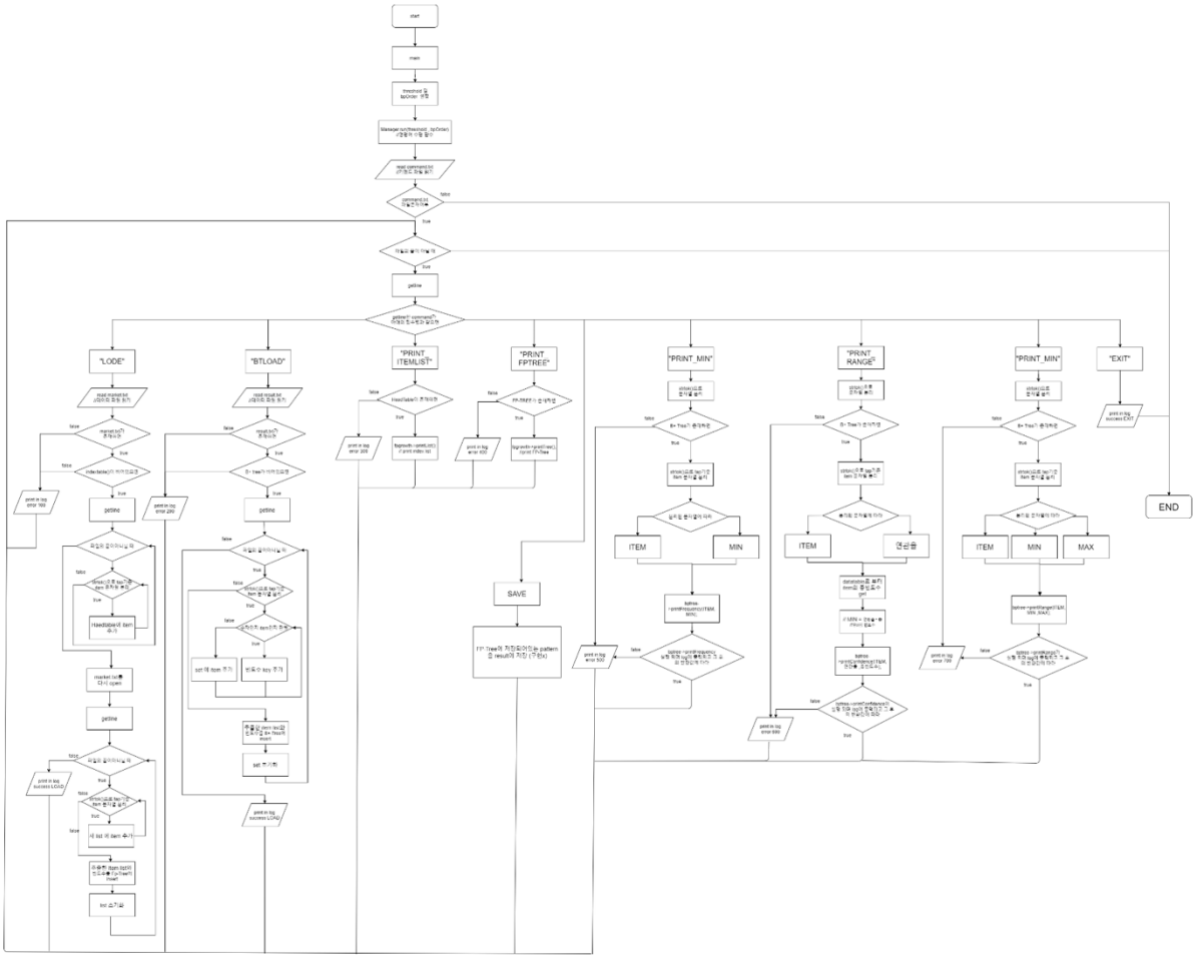


D. Class 설명과 각 멤버 변수와 함수

Class	멤버 함수, 변수	설명
Manager	FpGrowth * fpgrowth	Fp_growth class 동적할당 포인터
	BpTree* bptree	Bp tree의 class 동적할당 포인터
	Char* cmd, ifstream fin, ofstream flog	파일 읽고 쓰기, 명령어
	void run(const char * command)	Command txt에서 명령어 읽기
	LOAD, BTLOAD, PRINT_ITEMLIST, PRINT_FPTREE, PRINT_RANGE, PRINT_CONFIDENCE, PRINT_MIN	명령을 수행하는 함수들
	void PrintErrorCode(int num) void PrintSuccess()	Log.txt에 쓰기 성공 여부
HeaderTable	list<pair<int, string> > indexTable	Item 이름 빈도수 저장 stl
	map<string, FPNode*> dataTable	Item 이름과 fp tree node 연결 map stl
	void insertTable(char* item, int frequency)	Indextable과 datatable 데이터 삽입
	list<pair<int, string>> getindexTable()	Indextable의 주소 반환
	map<string, FPNode*> getdataTable()	Datatable의 주소 반환
	FPNode* getNode(string item)	Datatable의 연결된 노드 반환
	void descendingIndexTable() void ascendingIndexTable()	내림차순 또는 오름차순 정리
	int find_frequency (string item)	입력받은 item 이름의 빈도수 정리

FPNode	char* item	Node의 item 이름
	Int frequency	빈도수
	FPNnode* parent	부모노드 주소
	FPNode* next	다음 노드
	map<string,FPNode*>children	자식들의 노드 저장 map
FPGrowth	Int threshold	Threshold 기본값=3
	FPNode* fpTree	Fptree의 root 설정
	HeaderTable* table	Headertable의 동적할당 주소 저장
	map<set<string>, int> frequenctPatterns	Save 동작시
	fstream* fout ofstream flog	파일 읽고 쓰기
	void createFPtree(FPNode* root, HeaderTable* table, list<string> item_array, int frequency)	Fptree 생성 함수
	void connectNode(HeaderTable* table, string item, FPNode* node)	Fptree 생성할 때 각 node연결 함수
	bool printList()	Index list 출력
	bool printTree()	Bp tree 출력
BpTreeNode	BpTreeNode* pParent	공통적으로 사용하는 parent node
	pMostLeftChild = NULL	공통적으로 사용하는 child node
BpTreedataNode	Map<int,FrequentPatternNode*> mapData	Item pattern을 저장하는 node 저장, 빈도수 저장하는 map stl
	BpTreeNode* pNext BpTreeNode* pPrev	Datanode간 linked list 연결
FrequentPatternNode	Int frequency	빈도수
	Multimap<int,set<string>>FrequentPatternlist	패턴의 item 개수와 item pattern 저장 stl
BptreeindexNode	Map<int,BptreeNode*>mapindex	빈도수를 기준으로 정렬, 빈도수와 자식 node 저장하는 stl
Bptree	BpTreeNode* root	B+ tree의 root
	Int order	차수
	ofstream* fout	파일 읽고 쓰기
	bool Insert(int key, set<string> set)	B+ Tree에 삽입
	BpTreeNode* searchDataNode(int n)	Key 값을 이용하여 search 진행
	bool excessDataNode(BpTreeNode* pDataNode)	해당 data node가 차수를 넘어 overflow 발생 여부 확인
	bool excessIndexNode(BpTreeNode* pIndexNode)	해당 indexnode가 차수를 넘어 overflow 발생 여부 확인
	void splitDataNode(BpTreeNode* pDataNode)	Overflow 발생시 datanode 쪼개기
	void splitIndexNode(BpTreeNode* pIndexNode)	Overflow 발생시 indexnode Whrorl
	bool printFrequency(string item, int min_frequency)	최소 빈도수 이상, 연관율 이상, 최소빈도수와 최대빈도수 사이의 pattern을 출력하는 함수
	bool printConfidence(string item, double item_frequency, int min_frequency)	
	bool printRange(string item, int min, int max)	

2. Flowchart



코드 전체의 flowchart를 그리면 위와 같다. 각 명령어 별로 진행순서를 그렸으며 가산점인 SAVE 부분을 제외하고는 구현하였다.

3. Algorithm

A. Insert Fp-tree

- Fp-tree에 삽입하기 전에 header table을 완성한다.

1. 해당 라인 모든 물품에 접근하여 가져온다.
2. Headtalbe에서 item 빈도수가 threshold와 같거나 크면 삽입한다.
3. Root가 자식노드를 가지지 않은 경우 root에 자식노드를 추하가고 해당 item의 datatable의 pointer의 끝에 추가한다.
4. Root가 자식노드를 가지면 root부터 시작해서 해당 item과 이름이 같은 자식노드를 확인하고 해당 노드가 있으면 빈도수만 up 해준다. 만약 없는 경우 자식 노드를 새로 추가하고 headtable의 datatable의 pointer와 연결한다.
5. 다름 라인의 물품들에 접근하고 반복한다.

B. Print Fp-tree

1. Fp tree가 비어 있는 경우 오류 코드
2. 오름차순으로 정렬한 index table의 빈도수가 Threshold보다 크거나 같으면 물품을 가지고 data table의 pointer를 가지고 온다. 이때 이름과 빈도수를 {} 에 출력한다.

3. 해당 pointer의 다음 노드로 이동하여 ()안에 빈도수와 이름을 출력한다.
4. 부모 노드로 이동하여 위의 과정과 같이 출력하고 root에 도달할 때까지 3번을 반복하여 실행한다.
5. Pointer에 다음 노드가 NULL이 될때까지 3~4번 반복한다.
6. Pointer가 NULL이 되면 다음 index로 접근하고 2~4번 반복한다.

C. Search B+ tree
1. Currnode를 root로 위치시킨다.
2. Root의 mostleftchild가 null이면 해당 root를 반환한다.
3. Root의 mostleftchild가 null이 아니라면 입력된 n이 currnode의 맨 처음 kery값 보단 작으면 mostleftchild로 이동하고 아니라면 key 값보다 크거나 같으면 해당 key의 indexnode 포인터로 currnode를 이동한다. 그리고 마지막으로 key 값을 모두 비교한다.
4. 위 과정을 root의 mostleftchild가 null이 아닐 때까지 반복한다.
5. 위의 반복이 끝난 후 currnode를 반환하고 마친다.

D. Insert B+ tree
1. Root가 비어있다면 새로이 node를 생성하여 삽입한다.
2. Root에 데이터가 있으면 key 값을 삽입할 datanode를 search를 통해 찾는다.
3. 해당 위치에 key와 itempattern을 추가한다. 이때 datanode가 order보다 커져 overflow가 발생하면 datanode를 split 한다.
4. 만약 datanode의 부모가 존재하지 않으면 split하고 생성된 indexnode를 새로 할당하고 이때 data node가 root라면 새로 root를 변경해준다.
5. 만약 datanode의 부모가 존재하면 해당 부모 indexnode에 key와 pointer 모두 추가하고 이때 indexnode가 차수보다 커져서 overflow가 발생 시 indexnode를 명령어를 통해 split을 진행한다.
6. 만약 indexnode의 부모가 존재하지 않고 split하고 생성된 index node를 새로 할당하면 datanode가 root라면 새로 root를 변경한다.
7. 만약 indexnode의 부모가 존재하면 해당 부모 indexnode에 key와 pointer모두 추가하고 차수보다 또 커서 overflow가 발생하면 indexnode를 split을 해주는 작업을 반복한다.

E. Print B+tree
1. 빈도수 범위를 min와 max로 구분하여 진행한다.
2. 만약 root가 비어있다면 오류 코드를 출력한다.
3. Min을 삽입할 수 있는 datanode를 search하고 해당 datanode부터 물품을 포함한 frequent pattern을 검색한다.
4. 만약 발견한다면 Datanode 안에 접근하여 물품을 포함하고 key값이 min보다 작거나 max보다 크면 출력한다.
5. 다음 노드가 없을 때까지 data node를 이동하여 반복한다.

4. Result Screen
   A. Command.txt

```
LOAD
PRINT_ITEMLIST
PRINT_FPTREE
BTLOAD
PRINT_BPTREE whole wheat pasta 2
PRINT_CONFIDENCE soup 0.3
PRINT_RANGE mineral water 50 100
EXIT
```

명령어들을 한번씩 실행시켜 결과를 확인해 보았다.

   B. LOAD

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

Load 명령어를 통해 market.txt에 있는 것을 정상적으로 불러왔음을 확인하였다.

   C. PRINT_ITEMLIST

```
============PRINT_ITEMLIST============
Item    Frequency
mineral water  625
spaghetti  480
chocolate  447
eggs  436
french fries  432
green tea  409
```

Headtable에 저장되어 있는 itemlist를 빈도수가 가장 큰 순서대로 내림차순 정렬하여 출력하였다.

   D. PRINT_FPTREE

Testcase 1에서는 물품이 너무 적어서 testcase2로 했을 때 확실히 비교가 되었다.

```
============PRINT_FPTREE============
{StandardItem,Frequency}  (Path_Item,Frequency)
{napkins,3}
(napkins,1)(parmesan cheese,1)(low fat yogurt,1)(chocolate,52)(spaghetti,193)(mineral water,625)
(napkins,1)(shampoo,1)(gluten free bar,1)(tomato sauce,1)(muffins,1)(light mayo,1)(grated cheese,1)(frozen vegetabl
(mineral water,625)
(napkins,1)(carrots,1)(almonds,1)(red wine,1)(avocado,1)
{bramble,5}
(bramble,1)(cottage cheese,1)(soup,1)(frozen smoothie,2)(shrimp,4)(milk,20)(spaghetti,193)(mineral water,625)
(bramble,1)(red wine,1)(turkey,1)(eggs,90)(mineral water,625)
(bramble,1)(pepper,1)(grated cheese,1)(cake,4)(chocolate,187)
(bramble,1)(cauliflower,1)(gluten free bar,1)(vegetables mix,1)(energy bar,1)(salmon,1)(grated cheese,1)(escalope,1
12)(eggs,140)
(bramble,1)(fromage blanc,1)(fresh tuna,1)(cottage cheese,1)(honey,1)(escalope,1)(turkey,1)(green tea,11)(eggs,55)
(spaghetti,287)
{chutney,8}
(chutney,1)(yams,1)(protein bar,1)(energy drink,1)(whole wheat pasta,1)(cereals,1)(soup,1)(ground beef,15)(spaghett
(mineral water,625)
(eggs,140)
(eggs,90)(mineral water,625)
(eggs,55)(spaghetti,287)
(eggs,24)(chocolate,79)(spaghetti,287)
(eggs,12)(chocolate,52)(spaghetti,193)(mineral water,625)
{chocolate,447}
(chocolate,187)
(chocolate,128)(mineral water,625)
(chocolate,79)(spaghetti,287)
(chocolate,52)(spaghetti,193)(mineral water,625)
(chocolate,1)(chocolate,52)(spaghetti,193)(mineral water,625)
{spaghetti,480}
(spaghetti,193)(mineral water,625)
(spaghetti,287)
{mineral water,625}
(mineral water,625)
======================
```

   E. BTLOAD

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

정상적으로 출력이 되었다.

   F. Print_BPTREE

```
whole wheat pasta -> {antioxydant juice} 2
whole wheat pasta -> {bacon} 2
whole wheat pasta -> {frozen vegetables} 16
whole wheat pasta -> {eggs} 17
whole wheat pasta -> {mineral water} 17
whole wheat pasta -> {french fries} 20
whole wheat pasta -> {chocolate} 21
whole wheat pasta -> {spaghetti} 22
whole wheat pasta -> {milk} 30
============================
```

빈도수가 2보다 큰 경우 출력되도록 하였다.

G. PRINT_CONFIDENCE

```
soup -> {milk} 45 0.32
soup -> {spaghettl} 48 0.34
soup -> {mineral water} 76 0.54
==============================
```

연관률이 0.3 이상이라면 출력되도록 하였다.

H. PRINT_RANGE

```
mineral water -> {fresh bread} 50
mineral water -> {grated cheese} 52
mineral water -> {chocolate, spaghettl} 52
mineral water -> {escalope} 53
mineral water -> {salmon} 56
mineral water -> {turkey} 62
mineral water -> {frozen smoothie} 65
mineral water -> {whole wheat rice} 71
mineral water -> {low fat yogurt} 75
mineral water -> {soup} 76
mineral water -> {tomatoes} 76
mineral water -> {shrimp} 78
mineral water -> {chicken} 80
mineral water -> {burgers} 85
mineral water -> {cake} 98
============================
```

50보다 크고 100보다 작은 빈도수를 가진 pattern이 출력되오록 하였다.

5. Consideration

이번 프로젝트에서는 데이터 구조 설계 시간에 배운 Fp-growth와 B+ tree를 이용하여 물품 데이터를 읽고 이를 분석하는 프로그램을 설계하였다.

1차 과제가 단순 linked list였다면 이번엔 stl과 B+ tree와 같은 것들을 활용하여 각 class 또한 상속시켜 진행하다 보니 매우 복잡하였다. 다만 skeleton code가 매우 잘 잡혀 있어 과제를 진행하는 길라잡이가 되어 향후 본인이 구현해야 할 코드가 무엇인지 알려주는 역할을 해주었다.

B+ tree를 다 작성 후 command를 읽어오는 과정에서 각 물품을 본인이 tab으로 구분했음을 잊고 명령어는 단순 띄어쓰기로 가져오는 과정에서 오류를 겪었다. 하지만 이를 깨닫고 바로 수정하여 오류를 고칠 수 있었다.

STL의 경우 map 그리고 multimap 등을 사용하였고 구현하려는 자료구조에 맞게 stl을 사용하면 개발시간을 상당히 줄일 수 있음을 확인하였다. 다만 처음엔 이런 STL에 익숙하지 않았던 만큼 FP-tree 그리고 B+tree의 코드를 구현하는 것에 어려움을 겪었다. 하지만 구글링을 통해 각 STL을 어떻게 사용하는지 어떤식으로 구현하는 것이 좋을 지 탐색하며 begin그리고 rbegin 명령어들을 이용하여 간략히 코드를 작성할 수 있었다. 무엇보다 stl을 사용하면 명확하게 구현 의도를 표현할 수 있어서 향후 프로젝트에서도 stl을 최대한 활용할 수 있도록 무엇보다 효율적으로 사용할 수 있도록 노력해볼 것이다.