Data Structure Lab. Project #2

과목명	데이터구조실습
담당 교수	공진흥
학과	컴퓨터정보공학부
학번	2021202078
이름	최경정

1. Introduction

본 프로젝트는 FP-Growth와 B+Tree를 이용하여 상품의 빈도수를 저장해 최종적으로는 조합을 추천하는 프로그램을 구현하는 것을 목표로 한다. 이 프로그램은 장바구니 데이터에서 같이 구매한 상품들을 줄 단위로 받아 FP-Growth를 구현한다. 이때, FP-Tree는 상품별로 빈도수 및 상품의 이름, 상품과 관련이 있는 다른 FP-Tree의 상품들의 정보를 연결한다. 그리고 Header Table에는 상품 노드의 빈도수와 이름을 저장하며, FP-Tree의 다른 노드들과 연결되어 있다. FP-Growth를 만드는 과정에서 연결된 상품끼리 묶은 Frequent Pattern들은 Save 명령어를 통해 result.txt.에 빈도수, 상품 순으로 저장된다. 그리고 Frequent Pattern이 저장된 result.txt를 BTLOAD 명령어를 통해 빈도수를 기준으로 B+-Tree에 저장한다. B+-Tree는 IndexNode와 DataNode로 구성되는데, IndexNode는 DataNode를 찾기 위한 노드로써 존재하고, DataNode는 해당 빈도수를 가지는 Frequent Pattern이 저장되어 있는 Node이다. 프로젝트의 각 명령어에 대한 상세 설명은 다음과 같다.

1) LOAD

Market.txt 파일의 데이터 정보를 불러오는 명령어로, 파일에 데이터가 존재하는 경우 이를 읽어와 최종적으로 FP-Growth를 구현한다. Market 파일을 읽지 못하거나, 구현 후 FPTree를 불러오지 못한다면 에러 코드를 출력한다. 텍스트 파일에서 줄 단위로 읽어온 상품을 구분하여 HeaderTable을 생성하고, HeaderTable의 IndexTable에 상품의 빈도수와 상품명을, DataTable에 상품의 빈도수와 FpTree의 Pointer를 삽입한다. 그리고 headerTable를 바탕으로 하여 FPTree를 생성한다. FPTree에는 받아왔던 상품들 중 같이 구매했던 제품들은 연결되어 있으며, leafNode들은 next를 통해 dataTable과도 연결되어 있다. FPtree에는 threshold보다 큰 상품들만이 들어가 있다.

2) BTLOAD

Result.txt 파일의 데이터 정보를 불러오는 명령어로, 텍스트 파일에 데이터 정보가 존재하는 경우, 빈도수와 상품들을 읽어 b+-tree에 저장한다. 만약 텍스트 파일을 읽지 못하거나, 자료구조에 이미 데이터가 들어가 있는 경우 에러 코드를 출력한다. b+-tree는 IndexNode와 DataNode로 이루어져 있는데, IndexNode는 DataNode search를 위한 노드로 빈도수와 다른 FPNode(index node, data node)를 가리키는 포인터로 이루어져 있다. DataNode는 Frequentpattern이 저장된 노드로, FrequentPattern 안에는 집합 안에 존재하는 상품의 수와 상품명이 저장되어 있다. indexNode끼리와 indexnode, datanode 사이는 parent, child로 연결되어 있으며, datanode끼리는 next를 통해 연결되어 있다.

3) PRINT_ITEMLIST

FP-Growth의 HeaderTable에 저장된 상품들을 빈도수를 기준으로 내림차순으로 출력하는 명령어이다. IndexTable을 정렬 후 출력하는 방식을 사용하며, 이때 threshold보다 작은 빈도수를 가진 상품일지라도 전부 출력한다. 출력 형식은 상품명, 빈도수 이며, HeaderTable이 비어 있는 경우에는 에러 코드를 출력한다.

4) PRINT_FPTREE

FP-Growth의 FP-tree정보를 출력하는 명령어로, 출력 기준은 다음과 같다.

- Header Table의 오름차순 순으로 FP-Tree의 path를 출력
- threshold보다 작은 상품의 경으 출력하지 않음
- HeaderTable의 상품을 {상품명, 빈도수}의 형식으로 출력
- 해당 상품과 연결된 FP-Tree의 path들을 root 노드 전까지 연ㄹ결된 부모 노드들을 출력
- 해당 상품과 연결된 다음 노드듣ㄹ이 없을 때까지 출력하고, 다음 노드로 이동 시 다음 줄로 같이 이동

우선 Header Table을 빈도수를 기준으로 하여 오름차순으로 재정렬한다. Table의 DataTable을 탐색하며 threshold보다 큰 빈도수의 dataNode가 존재한다면 dataNode가 가리키는 노드로 이동하여 root까지 위로 올라가면서 출력하고, path 출력을 마친다면 next로 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 입력받는다. 만약 B+-Tree가 생성되어 있지 않거나 인자의 수가 다를 경우 error code를 출력한다. B+-Tree의 root부터 시작해 indexNode를 기준으로 search를 진행하며, 최소 빈도수 이상의 datanode를 만났을 경우 그 안의 set을 모두 출력한다.

8) PRINT_CONFIDENCE

B+-Tree에 저장된 Frequent Pattern을 출력하는 명령어이다. 그 중, 입력된 상품과 일정 연관율 이상의 confidence 값을 가지는 Frequent Pattern만을 출력하며, 연관율은 해당상품의 총 빈도수 분의 부분집합의 빈도수로 계산할 수 있다. 첫 번쨔 인자로는 상품명을 입력받고 두 번쨔 인자로는 연관율을 받는데, 이때 2개의 인자가 모두 입력되지 않거나 인자의 형식이 다를 경우 에러코드를 출력한다. 또한 B+-Tree가 비어 있거나, 탐색했는데 조건을 만족하는 dataNode가 없을 경우에도 에러 코드를 입력한다. 초기 인자 입력 후, 입력받은 연관율과 해당 상품의 총 빈도수의 곱보다 큰 빈도수를 B+-Tree에서 탐색한다. 탐색 종료 시 B+-Tree에 저장된 연관율 이상의 Frequent Patter을 출력하며 이동한다.

7) PRINT_RANGE

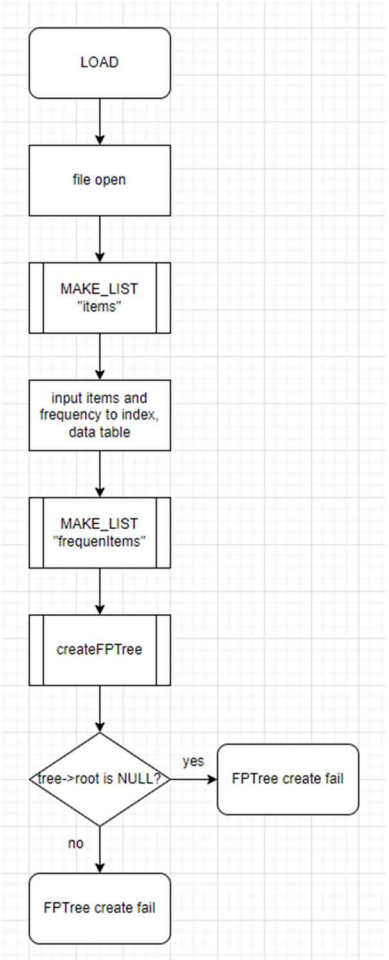
B+-Tree에 저장된 Frequent Pattern을 출력하는 명령어로, 첫 번쨔 인자로 상품명을 입력하고 두 번째 인자로는 최소 빈도수, 세 번째 인자로는 최대 빈도수를 입력받는다. PRINT_BPTREE와 유사한 기능을 하나, 최소 빈도수 이상 최대 빈도수 이하의 dataNode를 탐색해 Frequent Pattern을 출력하느다는 점이 다르다. 명령 실행 시, 세 개의 인자가 제대로 입력되지 않거나 B+-Tree가 비어 있는 경우, 탐색했는데 조건을 만족하는 dataNode가 없을 경우에는 에러 코드를 출력한다.

9) EXIT

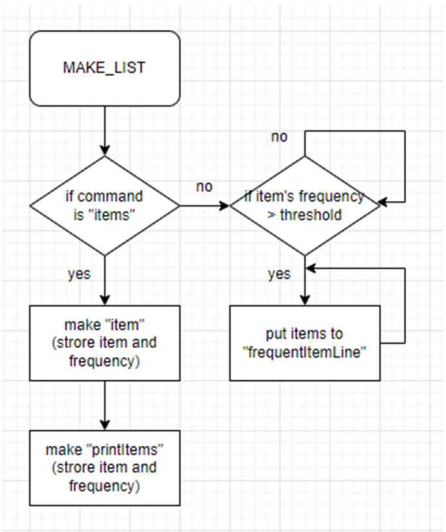
프로그램 상의 모든 메모리를 할당 해제하고, 프로그램을 종료한다.

2. Flowchart

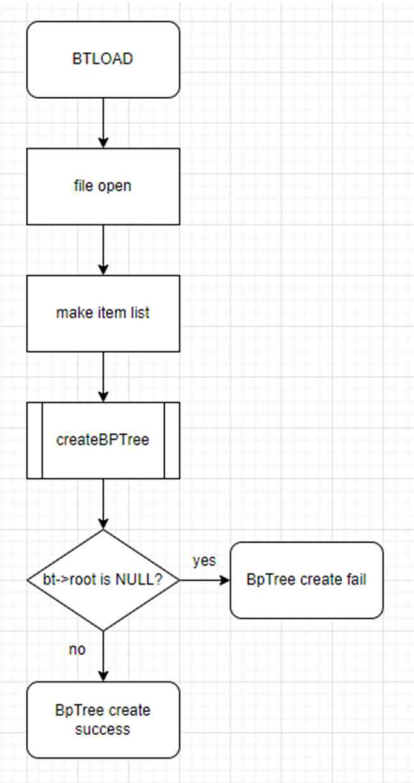
Manager.cpp에 저장된 명령어 함수들을 기준으로 하여 플로우 차트를 작성했다.
HeaderTable.cpp나 FPGrowth.cpp, BPTree.cpp에 있는 함수와 동작들은 Algorithm 파트에서 자세히 후술하도록 한다.



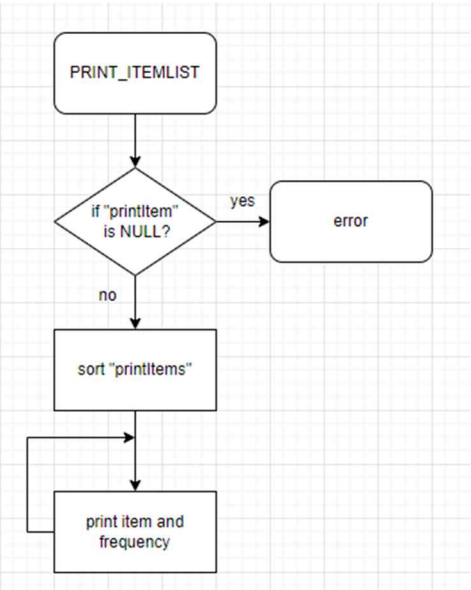
▲ LOAD 명령어 실행 시 수행되는 동작. HeaderTable을 생성하고, createFPTree 함수를 호출.



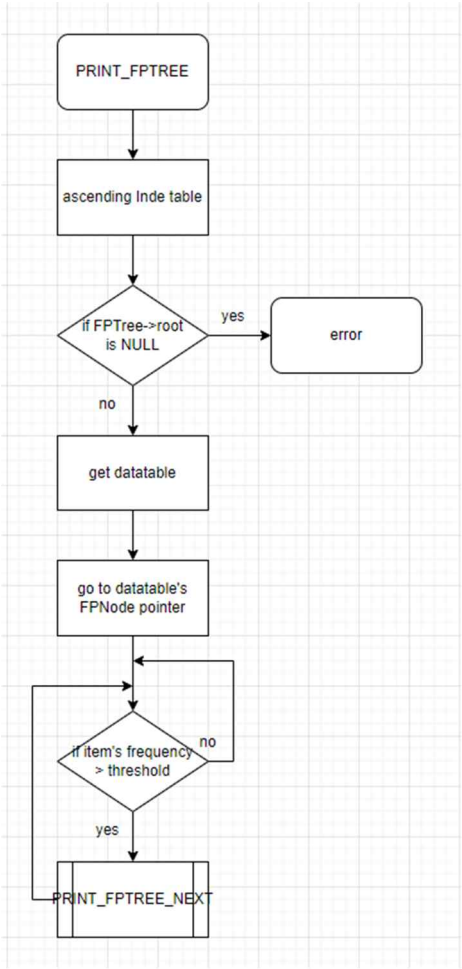
▲ MAKE_LIST 함수 실행 시 수행되는 동작. LOAD 명령어 중 사용되며, 여러 list 구조를 만드는 함수.



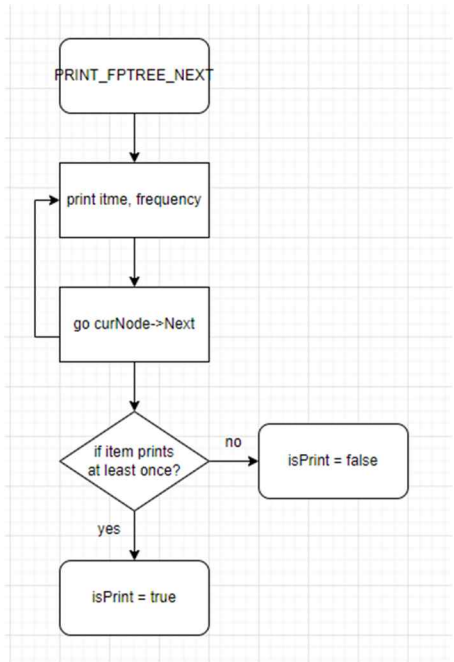
▲ BTLOAD 명령어 실행 시 수행되는 동작. BPTREE의 생성은 createBPTree 함수 호출을 통해 이뤄짐.



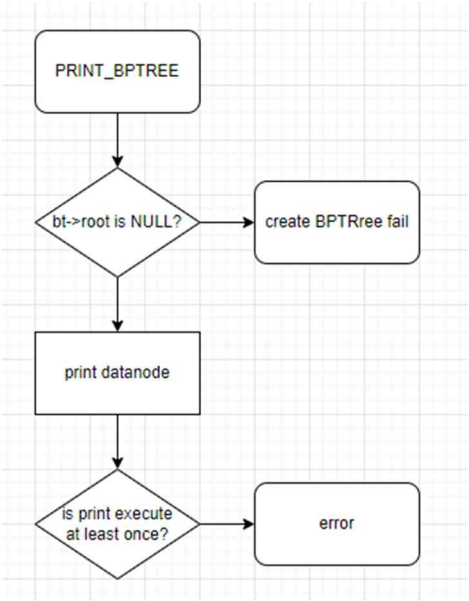
▲ PRINT_ITEMLIST 명령어 실행 시 수행되는 동작. MAKE_LIST 함수를 통해 사전에 만들어둔 list를 정렬 후 순회하며 item 및 frequency 출력.



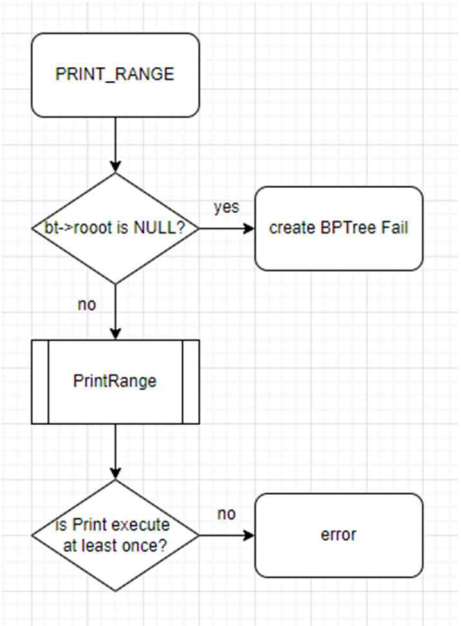
▲ PRINT_FPTREE 명령어 실행 시 수행되는 동작. headerTable 정렬 후 정렬된 dataTable을 순회하며 dataTable Node에 대해 PRINT_FPTREE_NEXT 함수를 호출.



▲ PRINT_FPTREE_NEXT 함수 실행 시 수행되는 동작. dataTable과 연결된 노드에서 root까지 parent를 타고 올라가며 출력 후, 다시 밑으로 내려와 next로 이동 후 출력 반복. PRINT_FPTREE 함수에서 호출되어 사용됨.



▲ PRINT_BPTREE 명령어 실행 시 수행되는 동작. indexNode를 적절히 활용하여 dataNode로 접근해 입력받은 최소빈도수 이상의 frequent pattern 출력.



▲ PRINT_RANGE 명령어 실행 시 수행되는 동작. indexNode를 적절히 활용하여 dataNode로 접근해 입력받은 최소빈도수 이상, 최대빈도수 이하의 frequent pattern 출력.

3. Algorithm

- createFPtree

1. 매개변수로 전달받은 string list item_array를 list의 시작부터 끝까지 순회함
1-2. parentNode를 root로 설정
2. newNode 생성 후 frequency를 1로 입력, 그리고 Next를 NULL로 입력
3. 만약 root의 child가 없을 경우, ParentNode를 root로 재설정
4. 아닐 경우, parentNode의 dataNode의 map container를 순회
5. 만약 순회한 map의 string 이름이 newNode의 string 이름과 같다면, 전자의 frequency를 1 증가
6. 순회한 map의 FPNode pointer를 parentNode로 설정 후 break
7. 만약 끝까지 다 돌았는데도 같은 이름의 string을 발견하지 못했을 경우, parentNode의 children에 newNode를 넣어줌
8. newNode의 parent로 parentNode를 설정
9. parentNode를 newNode로 설정
10. modifyDataTable 실행
11. 반복문을 매개변수로 받은 item_array가 끝날때까지 1~10 반복

- modifyDataTable

1. 매개변수로 받은 Node의 dataTable에 대하여 begin부터 순회 시작
2. 만약 순회하는 노드의 string 이름이 입력받은 item 이름과 같을 경우
2-1. 순회하는 노드의 FPNode pointer가 NULL이라면, 순회하는 노드가 가리킬 pointer에 매개변수로 받은 node를 설정 후 return
2-2. 아닌 경우, curNode를 순회하는 노드가 가리키는 FPNode로 설정 후 next로 이동할 수 있는 만큼 이동
2-3. curNode의 next를 매개변수로 받은 node로 설정 후 return

- find_frequency

1. frequency를 0으로 설정
2. indexTable을 begin부터 end까지 순회
3. 만약 순회하는 노드의 string 이름이 매개변수로 받은 item 이름과 동일할 경우
3-1. 순회하는 노드의 frequency를 frequency로 설정
4. frequency를 return

- connectNode

1. table에서 item이 있는 노드를 getNode통해 반환, 이를 curNode로 설정
2. curNode2를 curNode와 동일하게 설정
3. curNode2의 next가 없을 때까지 curNode2를 next로 이동

4. curNode2의 next를 매개변수로 받은 node로 설정

- Insert
1. freNode라는 FrequentpatternNode 생성 후 frequency와 list를 매개변수로 입력받은 key, set으로 설정
2. root가 NULL인 경우
2-1. newDataNode 생성 후 newDataNode의 DataMap에 key와 frequency 입력
2-2. root를 newDataNode로 설정 후 return
3. pCur을 searchDataNode를 통해 새 노드를 삽입할 자리로 설정
4. pCur의 DataMap의 begin부터 end까지 순회
4-1. 만약 순회하는 노드의 key값이 매개변수 key 값과 같은 경우, 순회하는 노드에 set list를 삽입하고 return
4-2. 만약 순회하는 노드의 key값이 매개변수 key값보다 큰 경우, 삽입할 곳을 찾은 것이므로 break
4-3. 둘 다 아닐 경우, 계속 순회를 진행
5. pCur의 DataMap에 key와 freNode를 삽입
6. 만약 order-1보타 DataMap의 size가 크다면, DataNode의 split 진행

- searchDataNode
1. DataNode로의 접근 위해 pCur을 root로 설정 후, MostLeftChild로 계속 이동
2. pCur의 DataMap을 begin부터 end까지 순회
3. 만약 순회하는 노드의 key값이 매개변수 값과 같다면, 현재 pCur을 return
4. 만약 순회하는 노드의 key값이 매개변수 값보다는 크고, 현재 순회 노드가 DataMap의 첫 map이라면 pPrev(pCur보다 하나 전의 노드) return
5. 만약 순회하는 노드의 key값이 매개변수 값보다는 크고, 현재 순회 노드가 DataMap의 첫 map은 아니라면, pCur을 return
6. 반복문이 끝날 때까지 key값과 일치하는 map을 찾지 못했다면, pCur과 pPrev를 next로 이동시킴
7. pCur이 마지막 노드에 도달했다면, pCur을 return

- splitDataNode
1. pDataNode는 split해야 하는 dataNode, frontNode는 split한 앞부분이 들어가는 dataNode, backNode는 split한 뒷부분이 들어가는 dataNode
1-1. 위로 올라가야 하는 Node부분 전까지의 map을 frontNode에 삽입
1-2. 위로 올라가야 하는 Node부분을 포함해 pDataNode의 map 끝까지를 backNode에 삽입
1-3. newNum으로 위로 올라가야 하는 node의 key 값 저장
2. indexNode가 존재하지 않을 경우
2-1. newIndexNode 생성 후 root를 newIndexNode로 설정

2-2. newIndexNode의 key값을 newNum으로, 가리키는 노드를 backnode로 설정
2-3. newIndexNode의 mostLeftChild를 frontNode로 설정
2-4. frontNode와 backNode의 parent를 모두 newIndexNode로 설정
3. pDataNode의 parentNode(기존 indexNode)에 key값이 newNum, 가리키는 노드가 backNode인 새 indexMap 삽입
4. 만약 frontNode의 첫 map의 key값이 pDataNode의 parent indexNode의 첫 key값보다 작을 경우, pDataNode의 parent indexNode의 mostLeftChilde를 frontNode로 설정
5. pDataNode의 parent indexNode의 begin부터 end까지 순회
5-1. 만약 frontNode의 DataMap의 key값이 pDataNode의 parent indexNode의 key값과 같다면, pDataNode의 parent indexNode의 가리킬 노드로 frontNode를 설정
5-2. 만약 새롭게 올라간 노드의 key값과 pDataNode의 parent indexNode의 key값이 같다면, pDataNode의 parent indexNode의 가리킬 노드로 backNode 설정
6. frontNode와 backNode의 Parent로 pDataNode의 parent indexNode 설정
7. 만약 pDataNode의 parent indexNode의 size가 order-1 를 넘는다면, indexNode split 진행
8. 기존 pDattaNode와 prev, next 관계였던 노드들을 각각 frontNode, backNode와 연결
9. frontNode와 backNode를 prev, next 관계로 연결

- splitIndexNode
1. pIndexNode는 split해야 하는 IndexNode, frontNode는 split한 앞부분이 들어가는 indexNode, backNode는 split한 뒷부분이 들어가는 indexNode
1-1. 위로 올라가야 하는 Node부분 전까지의 map을 frontNode에 삽입
1-2. 위로 올라가야 하는 Node부분을 미포함한 pIndexNode의 map 끝까지를 backNode에 삽입
1-3. newNum으로 위로 올라가야 하는 node의 key 값 저장
2. pIndexNode의 parent indexNode가 존재하지 않을 경우
2-1. newIndexNode 생성 후 root를 newIndexNode로 설정
2-2. newIndexNode의 mostLeftChilde로 frontNode 설정
2-3. frontNode와 backNode의 parent로 모두 newIndexNode 설정
2-4. frontNode의 mostLeftChild를 pIndexNode의 parent indexNode의 mostLeftChild 노드로 설정
3. pIndexNode의 parentNode(기존 indexNode)에 key값이 newNum, 가리키는 노드가 backNode인 새 indexMap 삽입
4. findIndexMap으로 pIndexNode의 parent indexNode 설정, fkey로 frontNode의 indexMap의 첫 key값을, bkey로 backNode의 indexMap의 첫 key값을 설정
5. 만약 순회하는 노드가 parentNode의 mostLeftChilde로 설정된 노드라면, pIndexNode의 parent indexNode의 MostLeftChild로 frontNode를 설정
6. findIndexMap의 begin부터 end까지 순회
6-1. 순회하는 노드가 findIndexNode의 end라면, 순회하는 노드가 가리킬 IndexNode로 backNode 설정
6-2. fkey가 순회하는 노드의 전 노드의 key값보다 작다면, 순회하는 노드의 전 노드가 가리킬

IndexNode로 frontNode를 설정
6-3. bkey가 순회하는 노드의 전 노드의 key값보다 작다면, 순회하는 노드의 전 노드가 가리킬 IndexNode로 backNode를 설정
7. frontNode와 backNode의 parentNode로 pIndexNode의 parent indexNode 설정
8. frontNode의 mostLeftChild로 pIndexNode의 mostLeftChild 설정
9. backNode의 mostLeftChild로 새로 올라간 노드의 IndexMap의 begin으로 설정
10. 만약 pIndexNode의 parent indexNode의 size가 order-1보다 작다면, splitIndexNode 재실행

4. RESULT SCREEN

각 명령어별로의 결과 화면을 정리했다. 모든 결과는 콘솔창이 아닌 log.txt에 출력된다. 보고서에는 깃허브에 올라온 market1.txt 파일과 result1.txt 파일을 이용했다. Command의 순서는 다음과 같다.

```
1   LOAD
2   PRINT_ITEMLIST
3   PRINT_FPTREE
4   BTLOAD
5   PRINT_BPTREE soup 2
6   PRINT_CONFIDENCE
7   PRINT_RANGE soup 2 3
8   SAVE
9   EXIT
```

1. LOAD

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

▲ LOAD에 성공한 모습. HeaderTable의 IndexTable, DataTable이 성공적으로 생성되고, 이를 통해 FPTREE가 정상적으로 구현되었을 때의 모습이다.

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

========PRINT_ITEMLIST========
ERROR 300
========================


========PRINT_FPTREE========
ERROR 400
========================
```

▲ 파일을 읽지 못했을 경우 error 코드를 출력한다. LOAD 실패 시 ITEM_LIST나 FPTREE도 생성할 수 없으므로 모두 error가 뜬다.

2. PRINT_ITEMLIST

```
========PRINT_ITEMLIST========
Item    Frequency
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
```

▲ threshold를 2로 설정했는데, threshold보다 작은 빈도수를 가진 아이템도 잘 출력됨을 확인할 수 있다.

3. PRINT_FPTREE

```
========PRINT_FPTREE========
{StandardItem, Frequency} (Path_Item, Frequency)
{almonds, 2}
(almonds, 1) (burgers, 3)
(almonds, 1) (soup, 1) (chocolate, 1) (ground beef, 1) (burgers, 1) (turkey, 2)
{black tea, 2}
(black tea, 1) (mineral water, 1) (spaghetti, 1) (fresh tuna, 1) (turkey, 2)
(black tea, 1) (energy bar, 1) (milk, 1) (mineral water, 1) (spaghetti, 1) (ground beef, 1)
{brownies, 2}
(brownies, 1) (hot dogs, 1) (french fries, 1) (avocado, 1) (soup, 4)
(brownies, 1) (chocolate, 1) (spaghetti, 3)
{escalope, 2}
(escalope, 1) (frozen smoothie, 1) (salmon, 1) (black tea, 1) (energy bar, 1) (milk, 1) (mineral water, 1) (spaghetti, 1) (
(escalope, 1) (frozen smoothie, 1) (whole wheat rice, 1) (honey, 1) (mineral water, 1) (spaghetti, 1) (frozen vegetables, 1
{fresh tuna, 2}
(fresh tuna, 1) (turkey, 2)
(fresh tuna, 1)
{frozen smoothie, 2}
(frozen smoothie, 1) (salmon, 1) (black tea, 1) (energy bar, 1) (milk, 1) (mineral water, 1) (spaghetti, 1) (ground beef, 1
(frozen smoothie, 1) (whole wheat rice, 1) (honey, 1) (mineral water, 1) (spaghetti, 1) (frozen vegetables, 1) (fresh tuna,
{frozen vegetables, 2}
(frozen vegetables, 1) (fresh tuna, 1)
(frozen vegetables, 1)
{grated cheese, 2}
(grated cheese, 1) (burgers, 3)
(grated cheese, 1)
{hot dogs, 2}
(hot dogs, 1) (french fries, 1) (avocado, 1) (soup, 4)
(hot dogs, 1) (eggs, 1) (almonds, 1) (soup, 1) (chocolate, 1) (ground beef, 1) (burgers, 1) (turkey, 2)
{pancakes, 2}
(pancakes, 1) (body spray, 1) (mineral water, 1) (spaghetti, 3)
(pancakes, 1) (body spray, 1) (brownies, 1) (hot dogs, 1) (french fries, 1) (avocado, 1) (soup, 4)
{pasta, 2}
(pasta, 1) (shrimp, 1)
(pasta, 1) (shrimp, 1) (grated cheese, 1) (burgers, 3)
{salmon, 2}
```

▲ standard item부터 시작해서 path를 통해 root까지 올라가는 모습을 확인해볼 수 있다. 겹치는 경로는 한 번만 출력되며, standard item의 빈도수보다는 작거나 같게 경로가 생성되어 있음을 확인할 수 있다.

4. BTLOAD

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

▲ BTLOAD에 성공한 모습. BpTREE가 정상적으로 구현되었을 때의 모습이다.

```
203    ========BTLOAD========
204    ERROR 200
205    ======================
206
207    ========PRINT_BPTREE========
208    ERROR 500
209    ======================
210
211
212    ========PRINT_RANGE========
213    ERROR 700
214    ======================
215
```

▲ 파일을 읽지 못했을 경우 error 코드를 출력한다. BTLOAD 실패 BpTREE를 생성할 수 없으므로 PRINT_BPTREE, PRINT_RANGE 모두 error가 뜬다.

5. PRINT_BPTREE

```
207    ========PRINT_BPTREE========
208    {almonds, soup} 2
209    {avocado, soup} 2
210    {body spray, soup} 2
211    {chicken, soup} 2
212    {frozen vegetables, soup} 2
213    {ground beef, soup} 2
214    {hot dogs, soup} 2
215    {milk, soup} 2
216    {mineral water, soup} 2
217    {protein bar, soup} 2
218    {almonds, burgers, soup} 2
219    {almonds, eggs, soup} 2
220    {body spray, green tea, soup} 2
221    {burgers, eggs, soup} 2
222    {burgers, french fries, soup} 2
223    {burgers, green tea, soup} 2
224    {chocolate, ground beef, soup} 2
225    {eggs, french fries, soup} 2
226    {frozen vegetables, soup, spaghetti} 2
227    {green tea, soup, spaghetti} 2
228    {milk, soup, spaghetti} 2
229    {almonds, burgers, eggs, soup} 2
230    {burgers, french fries, green tea, soup} 2
231    {burgers, soup} 3
232    {chocolate, eggs, soup} 3
233    {french fries, green tea, soup} 3
234    {chocolate, soup} 4
235    {eggs, soup} 4
236    {french fries, soup} 4
237    {soup, spaghetti} 4
238    {green tea, soup} 6
```

▲ 아이템을 soup, 최소 빈도수를 2로 지정했을 때의 모습

```
207    ========PRINT_BPTREE========
208    {burgers, soup} 3
209    {chocolate, eggs, soup} 3
210    {french fries, green tea, soup} 3
211    {chocolate, soup} 4
212    {eggs, soup} 4
213    {french fries, soup} 4
214    {soup, spaghetti} 4
215    {green tea, soup} 6
```

▲ 아이템을 soup, 최소 빈도수를 3으로 지정했을 때의 모습

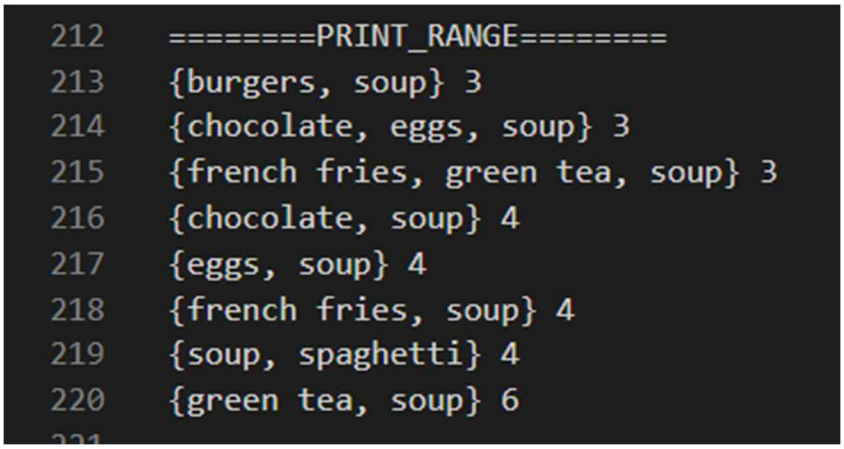
```
207    ========PRINT_BPTREE========
208    ERROR 500
209    ========================
```

▲ 아이템을 burger, 최소 빈도수를 5로 지정했을 때의 모습. 해당하는 FrequentPattern이 없기 때문에 에러코드를 띄운다.

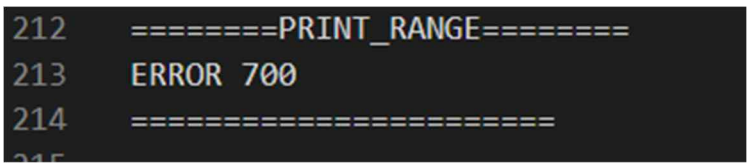
6. PRINT_RANGE

```
212    ========PRINT_RANGE========
213    {burgers, soup} 3
214    {chocolate, eggs, soup} 3
215    {french fries, green tea, soup} 3
216    {chocolate, soup} 4
217    {eggs, soup} 4
218    {french fries, soup} 4
219    {soup, spaghetti} 4
```

▲ 아이템을 soup, 최소 빈도수를 3, 최대 빈도수를 4로 설정했을 때의 모습



▲ 아이템을 soup, 최소 빈도수를 3, 최대 빈도수를 6으로 설정했을 때의 모습



▲ 아이템을 soup, 최소 빈도수를 7, 최대 빈도수를 8으로 설정했을 때의 모습. 해당하는 FrequentPattern이 없기 때문에 에러코드를 띄운다.

5. Consideration

우선 가장 해결이 어려웠던 문제는 BPTree에서 dataNode와 indexNode를 split하는 문제였다. dataNode split같은 경우는 고려해야 될 경우가 크게는 indexNode가 있을 때와 없을 때, 작게는 indexNode의 frontNode를 설정해줘야 할 지 말아야 할 지 정도의 문제였지만, index split의 경우에는 mostLeftChild 때문에 경우가 훨씬 많고 복잡하다고 생각했었다. 그러나 코드를 짜다 보니 세세한 부분이 너무 나뉘어지고 있다는 것을 발견했다. 그래서 묶을 수 있는 부분을 묶다 보니 경우의 수가 총 4개 정도로, dataNode의 split과 유사한 경우 수가 나왔다. 이를 통해 경우의 수를 세세하게 나누는 것이 다가 아니며, 공통된 부분은 묶어야 함을 깨달을 수 있었다.

또한 split을 진행하면서 답답했던 부분은, 내가 잘 하고 있는지를 확인하기가 어렵다는 것이었다. 저번 학기까지는 디버깅을 통해 변수들을 일일이 확인했지만, 이번에는 접근하는 곳이 너무 많고, 노드끼리 연결되고 상속되고 포함의 포함 관계가 반복되다 보니 내가 원하는 값을 디버깅으로 찾을 수 없는 지경에 이르렀다. 이때 생각을 바꿔서 print함수를 test용으로 하나 만들어 보았는데, 오히려 split 확인에 더 유리했다. 이를 통해 디버깅하는 것도 좋지만 너무 복잡한 경우 적당한 테스트용 출력 함수를 만드는 것이 더 확인하기 좋다는 것을 알 수 있었다.

그리고 지금까지는 디버깅을 진행할 때 계속 f10 버튼만을 눌러 왔는데, 이번에 너무 코드가 복잡해지다 보니 디버깅에 너무 많은 시간이 들어 디버깅에 대해 조금 더 찾아봤다. 그 결과 중단점의 제대로 된 의미와 f5의 기능에 대해서 알게 되었다.

또한 중간에 일정 부분 이상 파일을 읽을 수 없는 오류가 발생했는데, 이는 문자열 파싱에서 나온 문제점이었다. Result.txt 파일을 받을 때, 처음 문자열을 char*로 받았는데, 그러다 보니까 10의 경우 1로 받아지게 되고, 1의 노드로 계속 들어가게 되는 문제점이 생긴다는 것을 알게 되었다. 이를 해결하기 위해 char*로 받은 문자의 주소를 string형에다 받고, 그리고 stoi 함수를 통해 다시 int형으로 바꾸어 주었다. 이를 통해 stoi 함수에 대해 알게 되었고, 변환형에 대해서도 공부해보게 되는 계기가 된 것 같다.

중간에 자주 발생했던 문제 중 하나는 또 .과 ->의 차이 부분이었다. vscode에서는 어느 정도 잘못 입력하면 알아서 수정을 해 주지만, 해 주지 않은 경우에는 에러창이 떠서 수정해야 했던 경험이 몇 번 있었다. 변수 선언 시 . 를 사용하고, 포인터를 이용한 접근 시 ->를 이용하는 것 같은데 정확히 그 부분에 대해서는 나중에 조금 더 찾아보고 싶다.

아쉬움이 남았던 부분은 스켈레톤 코드의 활용에 대한 문제였다. 이미 주어진 코드 중에 이미 만들어진 함수가 있었는데, 그 부분을 모르고 새로 만들어서 쓴 경우도 있었고, 그리고 스켈레톤을 이해하지 못해 아예 처음부터 짠 부분도 있었다. 스켈레톤 코드의 틀을 활용하면 더 깔끔한 클린 코드를 짤 수 있었을거 같은데 그 부분에 대해서는 아쉬움이 남는다.

또한 프린트를 할 때 5중 for문을 사용하는 부분이 있는데, 그 부분이 아쉬움이 남는다. 성능 면에서 너무 좋지 않기 때문에 계속 신경이 쓰였다. 중간에 4중for문이 될 뻔 한 부분은 함수로 빼서 사용했는데, 함수로 빼면 사실상 기능은 동일하니까 이게 성능면에서 도움이 되기는 한 걸까? 라는 의문도 남아서 print함수 부분에서는 함수로 빼지 않고 사용했다. 클린 코드를 짜는 법은 나중에도 도움이 많이 될 것 같은데, 한 번 책을 찾아보고 연구해봐도 좋을 것 같다. 그리고 단순히 과제를 제출하는 데서 멈추지 않고 계속해서 코드를 수정하고 발전시켜 나간다면 이 또한 더 좋은 경험이 될 수 있을 것 같다. 그리고 1차에서와 마찬가지로 아직 비주얼 스튜디오 코드의 여러 기능에 대해서 잘 모르는데, 이를 조금 더 연구해 보면 나중에 3차 과제를 할 때나 학년이 올라갔을 때 더 편하게

코드를 작성할 수 있지 않을까 라는 생각도 든다.