

데이터 구조설계 및 실습

실험제목: Data Structure Lab. Project #2

제출일자: 2020 년 11 월 6 일 (금)

학 과: 컴퓨터정보공학부

담당교수: 이기훈 교수님

학 번: 2019202052

성 명: 김 호 성

1. Introduction

FP-Growth 와 B+-Tree 를 이용하여 상품 추천 프로그램을 구현한다.

- Program implementation

1. FP-Growth

- market.txt 에 저장된 데이터를 이용하여 구축한다.
- 한 번 장을 봤을 때 같이 산 물품은 w_n 으로, 각각의 물품은 w_t 루 구분된다.
- STL 라이브러리를 사용하여 구현한다.
- FP-Tree 와 Header Table 을 구현한다.
- Threshold 의 값은 2 이상으로 설정한다고 가정한다.
- FP-Growth 가 생성되면 result.txt 에 저장한다.

2. Header Table

- Header Table 은 Index Table 과 Data Table 로 구성한다.
- threshold 보다 작은 상품들도 저장한다.
- Index 는 빈도수를 기준으로 상품들이 정렬된 변수이다.
- Data 는 상품과 상품에 연결되는 노드를 가지고 있는 변수이다.
- Index 에서는 빈도수를 기준으로 오름차순과 내림차순으로 정렬을 할 수 있는 함수가 구현되어 있어야 한다.
- 정렬은 list 에서 제공되는 sort 함수를 이용하여 구현한다.

3. FP-Tree

- FP-Tree 클래스를 따로 생성하지 않고 FP Node 를 이용하여 구축한다.
- root 에서 자식 노드를 제외한 변수들은 NULL 값을 갖는다.

- 자식 노드들은 map 컨테이너 형태로 저장하며, 부모 노드를 가리키는 노드가 존재한다.
- key의 string은 상품명을 저장하고, value의 FP Node*은 해당 상품의 빈도수 정보 및 연결된 Node의 정보를 저장한다.
- FP-Tree에 저장된 노드들은 Header Table에서 같은 상품 노드들끼리 연결되어야 한다.
- FP-Tree에서 각 연결된 연관 상품에 따라 빈도수를 정확히 설정해야 한다.
- 연결되는 상품 순서는 빈도수를 기준으로 내림차순으로 결정된다.
- Header Table의 정렬 기준에 따라 결정이 된다.
- 저장되는 데이터는 상품명과 상품 노드이다.
- 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 "wt"을 구분자로 하여 연관된 상품들을 저장한 파일이다.
- B+-Tree는 인덱스 노드와 데이터 노드로 구성되며, 각 노드 클래스는 B+-tree 노드 클래스를 상속받는다.
- 데이터 노드는 단말 노드로, 해당 빈도수에 속하는 Frequent Pattern이 저장된 Frequent Pattern Node를 map 컨테이너 형태로 가지고 있으며 가장 왼쪽 자식을 가리키는 포인터를 따로 가진다.
- B+-Tree의 차수 ORDER(m)는 고정되어 있지 않으며 멤버 변수로 변경할 수 있다.

5. Frequent Pattern Node

- Frequent Pattern Node는 Frequent Pattern 정보를 multimap 컨테이너 형태로 가지고 있다.
- 각 map Data의 빈도수에 해당하는 Frequent Pattern의 정보를 저장하고 있어야 한다.

- key 로 Frequent Pattern 의 크기를 저장하고 value 로 Frequent Pattern 의 원소를 저장해야 한다.
- value 의 Frequent Pattern 의 원소는 STL 의 set 컨테이너 형태로 가지고 있다.
- set 에는 원소에 해당하는 상품명이 저장돼야 한다.
- Frequent Pattern 중 공집합 이거나 원소가 하나인 집합은 저장하지 않는다.

- Manager.cpp Introduction

run

- Command.txt 에서 함수를 읽어온 후 매치된 명령을 실행한다.
- 그 명령의 반환 값이 참이면 성공을, 거짓이면 실패를 log.txt 에 저장한다.

LOAD

- market.txt 에서 정보를 불러온다.
- 만약 텍스트 파일이 존재하지 않거나 자료구조에 이미 데이터가 들어가 있으면 에러 코드 100 을 출력한다.
- Insert Table 을 호출한 후 FP Tree 를 만든다.
- 상품명은 무조건 소문자로 주어진다.
- Wt 은 상품을, Wn 은 같이 구매한 물품을 구분한다.

BTLOAD

- result.txt 에서 정보를 불러온다.
- 텍스트 파일에 데이터 정보가 존재할 경우 파일을 읽어 B+-Tree 에 저장한다.
- 만약 텍스트 파일이 존재하지 않거나 자료구조에 이미 데이터가 들어가 있으면 에러 코드 200 을 출력한다.
- 첫 번째 값은 빈도수이고 그 이후 값은 상품 정보이다.

- wt 를 구분자로 하여 상품을 구분한다.
- Frequent Pattern 에 속한 상품들은 줄 단위로 구분된다.

PRINT_ITEMLIST

- FP-Growth 의 Header Table 에 저장된 상품들을 내림차순으로 출력한다.
- threshold 보다 작은 빈도수를 가진 상품도 출력한다.
- Header Table 이 비어 있는 경우 에러 코드 300 을 출력한다.

PRINT_FPTREE

- FP-Growth 의 FP-Tree 정보를 출력하는 명령어이다.
- Header Table 을 먼저 오름차순으로 정렬하고 threshold 이상의 상품들을 출력한다.
- FP-Tree 가 비어 있는 경우 에러코드 400 을 출력한다.

출력 조건

- Header Table 의 오름차순 순으로 FP-Tree 의 path 를 출력한다.
- threshold 보다 작은 상품은 넘어간다.
- Header Table 의 상품을 {상품명,빈도수}로 출력한다.
- 해당 상품과 연결된 FP-Tree 의 path 들을 (상품명,빈도수)로 root 노드 전까지 연결된 부모 노드를 출력한다.
- 해당 상품과 연결된 다음 노드들이 없을 때까지 출력하며 다음 노드로 이동하면 다음 줄로 이동한다.

PRINT_MIN

- B+-Tree 에 저장된 Frequent Pattern 중 입력된 상품과 두 번째 인자로 받은 최소 빈도수 이상의 값을 가지는 Frequent Pattern 을 출력하는 명령어이다.
- 첫 번째 인자로 상품명을 입력 받고, 두 번째 인자로 최소 빈도수를 받는다.

- 명령이 실행되면 입력 받은 최소 빈도수를 기준으로 B+-Tree 에서 탐색한다.
- 탐색이 끝나면 B+-Tree 에 저장된 Frequent Pattern 중 최소 빈도수 이상을 가지는 Frequent Pattern 을 출력하며 이동한다.
- 출력할 Frequent Pattern 이 없거나 B+-Tree 가 비어 있는 경우 에러 코드 500 을 출력한다.

PRINT_CONFIDENCE

- B+-Tree 에 저장된 Frequent Pattern 중 입력된 상품과 두 번째 인자로 받은 연관율 이상의 값을 가지는 Frequent Pattern 을 출력하는 명령어다.
- 첫 번째 인자로 상품명을 입력 받고 두 번째 인자로 연관율을 받는다.
- 연관율은 부분집합의 빈도수 / 해당 상품의 총 빈도수 로 계산이 된다.
- 명령이 실행되면 입력 받은 연관율과 해당 상품의 총 빈도수의 곱보다 큰 빈도수를 B+-Tree 에서 탐색한다.
- 탐색이 끝나면 B+-Tree 에 저장된 연관율 이상의 Frequent Pattern 을 출력하며 이동한다.
- 출력할 Frequent Pattern 이 없거나 B+-Tree 가 비어 있는 경우 에러 코드를 600 을 출력한다.

PRINT_RANGE

- B+-Tree 에 저장된 Frequent Pattern 을 출력하는 명령어다.
- 첫 번째 인자로 상품명을 입력하고 두 번째 인자로 최소 빈도수, 세 번째 인자로 상품명을 입력한다.
- 3 개의 인자가 모두 입력되지 않거나 형식이 다르면 에러 코드 700 을 출력한다.
- 명령이 실행되면 최소 빈도수를 가지고 B+-Tree 에서 Frequent Pattern 을 탐색한다. 탐색이 끝나면 최대 빈도수까지 B+-Tree 에 저장된 Frequent Pattern 을 출력하며 이동한다.

- 출력할 Frequent Pattern 을 출력하며 이동한다.
- 출력할 Frequent Pattern 이 없거나 B+-Tree 가 비어 있는 경우 에러 코드 700 을 출력한다.

SAVE

- FP-Growth 의 상품들의 연관성 결과를 저장하는 명령어다.
- 생성된 Frequent Pattern 들을 result.txt 에 저장한다. 저장 포맷은 'wt'를 구분자로 하여 빈도수, 상품명 순으로 저장한다. FP Growth 의 Frequent Pattern 이 비어 있는 경우 에러코드 800 을 출력한다.

EXIT

- 프로그램상의 메모리를 해제하며, 프로그램을 종료한다.

HeaderTable.h introduction

insertTable: Index Table 과 Data Table 을 구축한다.

getIndexTable: IndexTable 값을 반환한다.

getDataTable: DataTable 값을 반환한다.

getNode: DataTable 의 FPNode*를 반환한다.

descendingIndexTable: IndexTable 을 내림차순으로 정렬한다.

ascendingIndexTable: IndexTable 을 오름차순으로 정렬한다.

find_frequency: 특정 물품의 빈도수 값을 반환한다.

FPNode.h introduction

setParent: Parent 노드를 정해준다.

setNext: Next 노드를 정해준다.

Pushchildren: children 노드를 정해준다.

setItem: 노드의 물품명을 정해준다.

getItem: 노드의 물품명을 반환한다.

getNext: 노드의 next 노드를 반환한다.

getChildrenNode: 특정 물품에 children 노드가 존재하면 children 노드를, 존재하지 않는다면 NULL 을 반환한다.

getChildren: children 노드를 반환한다.

FPGrowth.h introduction

createTable: HeaderTable 에서 insert 함수를 호출하여 Header Table 을 만든다.

createFPtree: FPTree 를 구축한다.

connectNode: FPTree 의 노드와 Header Table 을 p.next 로 연결한다.

frequentPatternSetting: Header Table 을 오름차순한 후 frequent Pattern 값을 가져온다.

getFrequentPatterns: frequentPattern 을 반환한다.

powerSet: 입력된 data 의 부분집합을 구하고, 생성된 부분집합은 FrequentPattern 에 저장한다.

Contains_single_path: pnode의 자식이 없으면 true 를, 자식이 2 이상이면 false 를 반환하고, 1 이면 pNode 의 자식을 contains_single_path 로 확인한다.

Item_frequency: table 에서 특정 물품을 찾은 후 그 물품의 빈도 수를 반환한다.

getTree: fpTree 를 반환한다.

getHeaderTable: HeaderTable 을 반환한다.

printList: HeaderTable 을 출력한다.

printTree: FPTree 를 출력한다.

SaveFrequentPatterns: 빈번한 패턴을 result.txt 에 저장한다.

FrequentPatternNode.h introduction

setFrequency: frequency 값을 받는다.

InsertListL: FrequentPattern 을 multimap 형태로 구축한다.

getFrequency: Frequency 를 반환한다.

Getlist: FrequentPatternList 를 반환한다.

BpTreeNode.h introduction

setNext: 다음 노드를 연결한다.

setPrev 이전 노드를 연결한다.

getNext: 다음 노드를 반환한다.

getPrev: 이전 노드를 반환한다.

insertDataMap: BpTreeDataNode.h 에서 불러와 DataMap 을 구축한다.

insertIndexMap: BpTreeIndexNode.h 에서 불러와 IndexMap 을 구축한다.

deltemap: map 을 삭제한다.

getIndexMap: IndexMap 을 반환한다.

getDataMap: DataMap 을 반환한다.

BpTreeIndexNode.h introduction

insertIndexMap: IndexMap 을 삽입한다.

deleteMap: IndexMap 을 삭제한다.

getIndexMap: IndexMap 을 반환한다.

BpTreeDataNode.h introduction

setNext: 현재 노드의 Next 노드를 설정한다.

setPrev: 현재 노드의 Prev 노드를 설정한다.

getNext: 다음 노드를 반환한다.

getPrev: 이전 노드를 반환한다.

insertDataMap: mapData 를 생성한다.

deleteMap: mapData 를 삭제한다.

getDataMap: mapData 를 반환한다.

BpTree.h

Insert: 삽입이 성공하였는지 TRUE FALSE 로 반환한다.

excessDataNode: DataNode 를 excess 하는데 성공하였는지 TRUE FALSE 로 반환한다.

excessIndexNode: IndexNode excess 하는데 성공하였는지 TRUE FALSE 로 반환한다.

splitDataNode: DataNode 를 분할한다.

splitIndexNode: IndexNode 를 분할한다.

getRoot: Root 를 반환한다.

searchDataNode: key 값이 n 인 DataNode 를 검색한다.

printFrequentPatterns: Frequent Pattern 을 출력한다.

printFrequency: 물품의 빈도수를 출력한다.

printConfidence: 입력된 상품과 연관율 이상의 값을 가지는 Frequent Pattern 을 출력한다.

printRange: 특정 물품의 Frequent Pattern 중 범위에 맞는 빈도수만 출력한다.

2. Flowchart(의사코드) main

main

1. Manager manager(2,3)
2. manager.run("command.txt")

Run(manager.cpp 내에서)

1. log.txt 파일이 열렸는지 확인
2. 열렸다면 command 를 한 줄 단위로 입력 받음
3. 입력 받은 command 를 공백으로 다시 구분해줌.
4. 명령이 존재한다면 명령에 따른 함수 실행(Load, BTLoad...)
- 4-1 명령에 인자 값으로 토큰화하고 남은 command 도 같이 넣어 줌.

(후에 남은 명령들로 command 가 동작하는 게 달라짐. Ex) PRINT_MIN sooupt2)

LOAD

0. 이전에 LOAD 한 적이 있는지 검사함. (만약 LOAD 가 중복이라면 return false)
1. market.txt 파일이 열렸는지 확인 만약 실패하면 return 0; (false)
2. market.txt 에서 한 줄 단위로 data 를 가져옴.
3. 가져온 data 를 w_t 에 따라 토큰화 하면서 getHeaderTable()->insertTable 에 int 1 값과 함께 넣어 줌.
4. 열었던 파일의 버퍼를 클리어하고, 파일의 가장 앞으로 다시 돌아옴.
5. 한 줄 단위로 다시 값을 입력 받고, 각각의 토큰화 된 아이템들을 list 에 순서대로 넣어 줌.
6. 만든 list 를 createFPtree 의 인자값으로 넣어 줌.
7. LOADCHECK = 1 로 하고 true 를 반환함.

BTLOAD

구현되지 않음

PRINT_ITEMLIST

0. flog 를 사용하여 출력 구조에 맞게 만듦.

1. fpgrowth 에 존재하는 printList 함수를 호출함.

1-1. table->getIndexTable()함수를 사용하여 indexTable 을 가져옴.

1-2. table 의 size 가 1 보다 작으면 false 값을 반환함.

1-3. table 의 size 가 0 일때까지 반복하여 가장 앞에 값의 item 과 frequency 를 출력

1-3-1. 출력 후 가장 앞의 값 삭제(자동으로 두 번째 값이 가장 앞에 값

1-4. 반복문이 정상적으로 종료된 후 true 값을 반환함/

PRINT_FPTREE

구현되지 않음

PRINT_MIN

구현되지 않음

PRINT_CONFIDENCE

구현되지 않음

PRINT_RANGE

구현되지 않음

SAVE

구현되지 않음

EXIT

1. 무조건 successCode 출력 후 run 반복문 탈출
2. 동적할당한 메모리들 close
3. return

Else (not match command)

1. Command Error 출력

3. Algorithm

LIST

시퀀스 컨테이너의 일종으로 순서를 유지하는 노드 기반 컨테이너다.

이중 연결 리스트이다.

멤버함수

`l.push_back(x)` // l의 끝에 x를 추가(원소가 추가됨, size 늘어남)

`l.pop_back()` // l의 마지막 원소 제거

`l.pop_front()` // l의 첫번째 원소 제거

`l.begin()` // l의 첫 원소를 가리키는 반복자

`l.end()` // l의 끝을 표시하는 반복자

`l.merge(l2)` // l2를 l로 합병 정렬한다.

`l.size()` // 원소의 개수(모든 컨테이너에서 제공)

`l.resize(n)` // l의 크기를 n으로 변경 (resize 또는 rsize)

`l.sort()` // l의 모든 원소를 오름차순으로 정렬

`l.splice(p,l2)` // p가 가리키는 위치에 l2의 모든 원소를 잘라 붙임

`l.front()` // l의 첫 번째 원소 참조

`l.back()` // l의 마지막 원소 참조

`l.empty()` // l가 비었는지 확인

MAP

연관 컨테이너의 일종으로 원소를 key와 value의 쌍으로 저장한다.

Set은 원소로 key 하나만을 저장하지만, map은 원소로 key와 value의 쌍을 저장한다.

연관 컨테이너에서 유일하게 [] 연산자 중복을 제공한다.

중복된 key 값을 갖지 않음

멤버함수

`m.insert(x)` // m 에 x 를 삽입

`m.begin()` // m 의 첫번째 원소의 반복자를 반환

`m.end()` // m 의 마지막 원소 다음의 반복자를 반환

`m.empty()` // m 이 비었는지 확인

`m.find(key)` // m 에서 key 를 찾아 해당 위치의 반복자를 반환

`m.erase(x)` // m 의 x 원소를 모두 제거

`m.swap(m2)` // m 과 m2 의 원소를 바꿈

`m.lower_bound(key)` // m 이 key 를 가지고 있다면 해당 위치의 반복자를 반환

`m.upper_bound(key)` // m 이 key 를 가지고 있다면 해당 위치 다음의 반복자를 반환

`m.clear()` // m 이 저장하고 있는 모든 원소를 삭제

`m.size()` // m 이 가지고 있는 원소의 개수를 반환

MULTIMAP

연관 컨테이너의 일종으로 기본적인 기능은 map 과 동일하다.

[] 연산자를 사용할 수 없다.

중복되는 값의 데이터를 저장할 수 있다.

멤버함수

`mm.insert(x)` // mm 에 x 를 삽입

`mm.begin()` // mm 의 첫번째 원소의 반복자를 반환

`mm.end()` // mm 의 마지막 원소 다음의 반복자를 반환

`mm.empty()` // mm 이 비었는지 확인

`mm.find(key)` // mm 에서 key 를 찾아 해당 위치의 반복자를 반환

`mm.erase(x)` // mm 의 x 원소를 모두 제거

mm.swap(m2) // mm 과 m2 의 원소를 바꿈

mm.lower_bound(key)// mm 이 key 를 가지고 있다면 해당 위치의 반복자를 반환

mm.upper_bound(key)// mm 이 key 를 가지고 있다면 해당 위치 다음의 반복자를 반환

mm.clear() // mm 이 저장하고 있는 모든 원소를 삭제

mm.size() // mm 이 가지고 있는 원소의 개수를 반환

mm.equal_range(key) // mm 이 가지고 있는 key 값의 범위를 반복자의 쌍으로 반환

FP-Growth

Header Table 과 FP Tree 로 구성된 알고리즘이다.

Header Table: 아이템 정보와 아이템의 빈도 수, FP Tree Node 와 연결이 저장되어 있는 Table 로 다시 Index Table(item, frequency)와 Data Table(item, FP Node*)로 나눌 수 있다.

- Data Table: 상품 이름과 연결된 FP Tree 노드를 저장하고 있는 테이블

- Index Table: 상품 이름과 빈도수를 저장하고 있는 테이블

FP Tree: root 는 자식 노드를 제외한 변수들은 NULL 값을 가지고 자식 노드를 map 컨테이너 형태로 가지고 있다.

또한, FP Growth 에는 threshold 가 존재하는데, FPNode 중 threshold 보다 값이 적은 frequency 를 가진 node 들은 Tree 와 Table 을 구성할 때 무시할 수 있다.

Frequent pattern

threshold 이상의 빈도수를 갖는 pattern 이다.

Header Table 에서 Threshold 조건에 맞는 아이템들 중 최하의 빈도수를 가지는 아이템부터 Frequent Pattern 을 찾는다.

또, single path 인지 확인해야 하는데 여기서 single path 란 FP-Tree 의 path 에 있는 노드에서 마지막 노드는 자식 노드가 없고 나머지 노드들은 자식 노드가 하나씩만 존재하는 경우를 말한다.

B+-Tree

B-tree 의 단점인 순회 작업을 보완하기 위해 고안된 트리로 단말 노드와 단말 노드가 아닌 것으로 나뉘어져 있다.

단말 노드: 데이터가 저장되어 있으며, 연결 리스트의 형태로 서로 연결되어 있음

= data Node

비단말 노드: 데이터의 빠른 접근을 위한 인덱스가 저장되어 있음

= index Node

4. Result Screen

Testcase1 로 돌렸을 때 log.txt

=====LOAD=====

Success

=====

===== ERROR 100 =====

=====

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

soda 1

shampoo 1

shallot 1

red wine 1

pet food 1

pepper 1

parmesan cheese 1

meatballs 1

ham 1

gums 1

fresh bread 1

extra dark chocolate 1

energy drink 1

cottage cheese 1

cookies 1

carrots 1

bug spray 1

unimplemented PRINT_FPTREE

===== ERROR 400 =====

=====

unimplemented SAVE

===== ERROR 800 =====

=====

unimplemented BTLOAD

===== ERROR 200 =====

=====

unimplemented BTLOAD

===== ERROR 200 =====

=====

unimplemented PRINT_MIN

===== ERROR 500 =====

=====

unimplemented PRINT_CONFIDENCE

=====PRINT_CONFIDENCE=====

Success

=====

unimplemented PRINT_RANGE

=====PRINT_RANGE=====

Success

=====

=====EXIT=====

Success

=====

5. Consideration

- testcase1 을 돌릴 때 Command.txt 의 정보는 다음과 같다.

LOAD

LOAD

PRINT_ITEMLIST

PRINT_FPTREE

SAVE

BTLOAD

BTLOAD

PRINT_MIN soup 2

PRINT_CONFIDENCE soup 0.3

PRINT_RANGE soup 3 5

EXIT

- 제출당시의 상태

Manager 에서의 명령어 핸들링: O

Header Table

- Index Table: O

- Data Table: O

FP-Growth

- Header Table: O

- FPNode: O

- 두 개의 연결: 시도는 하였으나 실패

이하 시도하지 못함.