Programming Assignment #1

학번: 2017041930

이름: 김선동

Index

1. Environment

OS는 linux 18.04, language는 c++를 사용하였다.

주석을 전부 한글로 달아 놓았기 때문에, 주석을 보려면 한글팩이 정상적으로 깔려 있어야 합니다.

2. Instructions for compiling my source codes

내가 사용한 컴파일러는 다음과 같다.

```
ubuntu@ubuntu:~/Assignment1$ g++ --version
g++ (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0
```

아래와 같이 컴파일하면 apriori.exe 실행파일을 얻을 수 있다.

```
ubuntu@ubuntu:~/Assignment1$ g++ -o apriori.exe main.cpp
```

또는 makefile을 첨부하였으니 make 명령어만으로 실행 파일을 생성할 수 있다.

```
ubuntu@ubuntu:~/Assignment1$ make
g++ -c -o main.o main.cpp
g++ -o apriori.exe main.o
```

아래와 같이 실행하면 output.txt에 정상적으로 결과물을 얻을 수 있다.

```
ubuntu@ubuntu:~/Assignment1$ ./apriori.exe 5 input.txt output.txt
```

3. Summary of my implementation

```
int main(int argc, char** argv)
{
    // ================================================================================
    // 실행은 3 개의 arguments가 주어진다. 3 개가 아니면 에러 문구 출력하고 종료한다.
    // 3 Arguments = {Minimum support, Input file name, Output file name}
    // Ex) Minimum support = 5%, Input file name = 'input.txt', Output file name = 'output.txt'
    // ================================================================================
    if (argc != 4)
    {
        cout << "다음 3 개의 arguments를 순서대로 입력해주세요. Minimum support, Input file name, Output file name" << "₩n";
        return -1;
    }
    ifstream readFile;
    ofstream writeFile;
    double minimumSupport = stod(argv[1]) / 100; // 백분율 기준으로 입력하기 때문에 100으로 나눠줘야 함
    string inputFileName = argv[2];
    string outputFileName = argv[3];

    printArgv(argv);
```

첫째로, main문에서 요구사항에 맞게 3개의 인자들을 입력 받는다. 3 개의 인자가 아니라면 에러 문구를 출력하고 종료한다. Minimum support와 input file name, output file name을 차례로 입력 받는다. Minimum support는 단위가 %이므로 100으로 나누어 준다. printArgv함수는 프로그램의 시작을 알리는 함수로 입력 받은 인자들을 출력해준다.

```
// ========================================
// input.txt 열어서 item_id의 set을 얻는다.
// 얻은 정보를 DB에 저장
// ========================================
readFile.open(inputFileName);
if (readFile.is_open())
{
    while (!readFile.eof())
    {
        string line;
        getline(readFile, line); // 각 Xact마다 중복된 item은 input으로 주어지지 않는다.
        if (line == "") break;   // 마지막이 개행으로 끝나기 때문에 input이 공백이면 break.

        int idx = 0;
        set<int> Xact;           // 중복된 item 있을수도 있으니 set으로 input 받는다. (오름차순)
        for (int pos = 0; pos < line.length(); pos++)
        {
            if (line[pos] == '₩t')
            {
                Xact.insert(stoi(line.substr(idx, pos - idx + 1)));
                idx = pos + 1;
            }
        }
        Xact.insert(stoi(line.substr(idx)));
        DB.push_back(Xact);      // input.txt안의 정보들 중 '₩t'없애서 item_id만을 담는 DB
    }
    readFile.close();
}
```

이제 입력 받은 input file name으로 파일을 열어서 eof까지 읽어주면서 정보를

저장해준다. 혹시 transaction에 중복된 item이 있을 수도 있으니 자료구조로 set을 선택했다. 이 set을 vector에 저장해준 것이 DB이고 말 그대로 input.txt의 모든 정보를 담고 있다. 이제 입력 파일의 모든 정보를 저장해주었으니 수도 코드를 따라 코딩해주면 된다.

```
// ============================================================================
// ****************************************************************************
// ******************                                      ******************
// ******************      The Apriori Algorithm Pseudo-Code      ******************
// ******************                                      ******************
// ****************************************************************************
// ============================================================================
// Initially, scan DB once to get frequent 1-itemset
// Generate candidate itemsets of length(k + 1) from frequent itemsets of length k
// Test the candidates against DB
// Terminate when no frequent or candidate set can be generated
//
// ********************************* Pseudo-code *********************************
//
// Ck: Candidate itemset of size k
// Lk: frequent itemset of size k
// L1 = {frequent items}
// for(k=1; Lk != empty; k++) do begin
//      Ck+1 = genCandidates(Lk);
//      for each Xact t in DB do
//          for each candidate c in Ck do
//              if c is contained in t then
//                  count++;
//          end
//      end
//      Lk+1 = Candidates in Ck+1 with min_support
// end
// return UkLk;
// ============================================================================
```

위 수도 코드는 강의 자료에도 나와 있는 코드로 length 별로 frequent itemset을 구하는 것이 목표이다. 우선 우리는 L1에 해당하는 1개짜리 itemset의 set을 구해야 한다. 이를 수행하는 것이 initialScan이라는 함수이다.

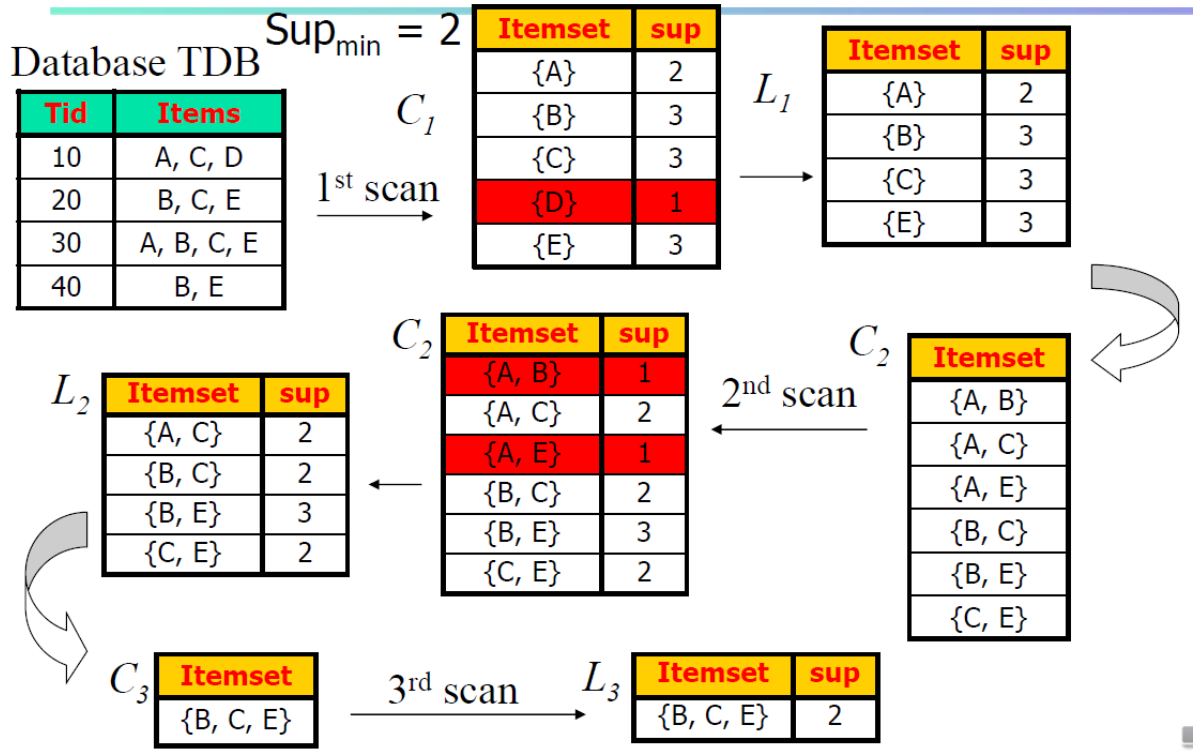
```
// ============================================================================
// L: k-length의 Frequent itemset과 그에 따른 sup을 저장하는 vector
// 최초에 DB를 스캔하며 frequent한 1-itemset 얻어서 L에 저장.
// ============================================================================
vector<map_itemset_sup> L = initialScan(minimumSupport);
```

```
// ========================================
// **************Initial Scan**************
// DB를 스캔하며 frequent한 1-itemset 얻는다.
// ========================================
vector<map_itemset_sup> initialScan(double min_support)
{
    vector<map_itemset_sup> L(2);
    for (auto& itemset : DB)
    {
        for (auto& item : itemset)
        {
            set<int> frequent_1_itemset;    // 1개 아이템을 지닌 아이템 셋 저장을 위한 set
            int item_id = item;
            int sup = 0;
            frequent_1_itemset.insert(item_id);

            if (L[1].find(frequent_1_itemset) != L[1].end())
                continue;

            for (auto& is : DB)             // DB scan하면서 sup 저장
                if (is.find(item) != is.end())
                    sup++;

            double cur_sup = (double)sup / DB.size();
            if (cur_sup >= min_support)
                L[1].insert(itemset_sup(frequent_1_itemset, sup));
        }
    }
    return L;
}
```

L은 itemset과 sup을 map으로 저장한 것을 원소로 갖는 벡터이다. 즉 L[1]은 length가 1이면서 frequent한 itemset을 가리키고 L[k]는 length가 k이면서 frequent한 itemset을 가리킨다. DB를 순회하면서 1개짜리 아이템셋을 찾고, 해당 아이템 셋의 support가 minimum support이상이라면 L[1]에 추가해준다.

```
// ========================================
// Frequent한 길이 k의 itemset으로 부터
// 길이 k+1의 candidate itemset 생성한다.
// 더이상 candidate 생성되지 않으면 종료
// ========================================
long long length = 1;
for (int k = 2; !L[length].empty(); k++)
{
    print123();
    candidates Ck = genCandidates(L[k - 1]);
    map_itemset_sup tmp;
    for (auto& itemset : DB)
    {
        for (auto& candidate : Ck)
        {
            if (isSubset(itemset, candidate))
            {
                if (tmp.find(candidate) == tmp.end())
                    tmp.insert({ candidate,1 });
                else
                    tmp[candidate]++;
            }
        }
    }
    // ============================================
    // DB를 순회하면서 생성한 앞서 candidate들 테스트
    // Minimum support 충족하지 못하면 drop하고
    // 충족한다면 L[k]에 추가한다.
    // ============================================
    map_itemset_sup::iterator it = tmp.begin();
    while (it != tmp.end())
    {
        double cur_sup = (double)it->second / DB.size();
        if (cur_sup < minimumSupport)
            tmp.erase(it++->first);
        else
            it++;
    }
    L.push_back(tmp);
    length++;
}
length--;
```

이제 L[k]가 더 이상 생성되지 않을 때까지 k를 늘려주면서 frequent한 itemset을 구해주어야 한다.



위와 같은 순서대로, candidate를 생성하고 sup을 검토해서 기준치 미달이면 drop하고 나머지는 L[k]에 추가해주면 된다.

```cpp
// ============================================================
// ********생성한 candidate가 적합한 지 확인하는 함수********
// 생성한 candidate가 frequent하려면
// 그의 부분 집합은 무조건 frequent하므로, L에 존재해야 한다.
// ============================================================
bool chkItemset(map_itemset_sup Lk, set<int> candidate)
{
    for (auto item : candidate)
    {
        set<int> chk_itemset(candidate);
        chk_itemset.erase(item);

        if (Lk.find(chk_itemset) == Lk.end())
            return false;
    }
    return true;
}

// ========================================
// ********Generate Candidates from Lk********
// Frequent한 길이 k의 itemset으로 부터
// k+1 길이의 Candidates 생성
// ========================================
candidates genCandidates(map_itemset_sup Lk)
{
    candidates Ck;
    for (auto i : Lk)
    {
        for (auto j : Lk)
        {
            if (i == j)
                continue;
            set<int> candidate(i.first);
            candidate.insert(j.first.begin(), j.first.end());

            long long k = i.first.size();
            if (candidate.size() == k + 1)
            {
                if (chkItemset(Lk, candidate))
                    Ck.insert(candidate);
            }
        }
    }
    return Ck;
}
```

위 함수는 앞서 이야기한 candidate를 생성하는 함수이다. Candidates는 set안에 set을 갖는 자료구조이다. L[k]를 돌면서 self joining을 해주고, k+1의 길이가 완성되었다면 frequent한 지 검사한다. Frequent한 지 검사하는 함수는 chkItemset으로 검사하는 itemset의 부분집합(길이 k의 itemset만 해당)이 L[k]에 존재한다면 true를 반환하고 존재하지 않는다면 false를 반환한다. 이 과정이 pruning에 해당된다.

```
// ========================================
// *********Subset인 지 확인하는 함수*********
// candidates들의 sup을 계산하기 위한 함수
// ========================================
bool isSubset(set<int> parent, set<int> pattern)
{
    // pattern과 parent 일치하면 is subset
    if (parent == pattern)
        return true;
    // pattern이 parent보다 길면 is not subset
    else if (parent.size() < pattern.size())
        return false;
    // pattern 안의 item이 parent에 없다면 is not subset
    else
    {
        for (auto item : pattern)
            if (parent.find(item) == parent.end())
                return false;
        return true;
    }
}
```

이렇게 생성된 candidates의 sup을 계산해주어서 minimum support를 충족하는지 검사해야 한다. 다시 DB를 순회하면서 생성한 candidate가 속해 있는 지 확인해야 한다. 이를 확인하는 함수가 isSubset이라는 함수이다. DB의 itemset이 parent가 되고, candidates들 중 각각이 pattern이 되어서 안에 속하는 지 확인한다. 일치한다면 true를 반환하고, pattern이 parent보다 길다면 false, 짧다면 일일

이 검사해서 확인해준다. 이를 통해서 true인 것들 만을 저장해준다. 이제 적합한 candidate와 그에 따른 sup을 저장해주었으니, minimum support를 충족하는 지 확인해주면 된다. 충족하지 않는다면 drop해주고 남은 것들을 L에 추가해준다. 각 for문마다 length를 늘려주어서 더 이상 생성되지 않았다면 반복문을 종료해주면 된다.

```
// ============================================
// 모든 frequent itemset을 map에 저장
// ============================================
for (auto& itemsets : L)
    for (auto& itemset : itemsets)
            all_frequent_set.insert(itemset);

writeFile.open(outputFileName);
writeFile << fixed << setprecision(2);

for (auto& itemsets : L)
{
    map_itemset_sup all_L;
    for (auto& itemset : itemsets)
        all_L.insert(itemset);

    // ============================================
    // Association Rules 생성
    // ============================================
    vector<information> outputVector = find_Association_Rules(all_L);

    // ============================================
    // 얻은 정보를 바탕으로 support와 confidence 계산
    // 출력 파일에 저장하기
    // ============================================
    write_Output(outputVector, writeFile, outputFileName);
}

writeFile.close();

cout << "₩n저장을 끝마쳤습니다.₩n";
```

Apriori algorithm을 통해 구한 모든 frequent한 itemset을 바탕으로 association rule을 구해야 한다. 나중에 confidence를 구하기 위해 L을 하나의 map에 모두

저장해준다. 주어진 요구사항에 맞게 소수점 2자리까지 반올림해서 표시하게 한다. 또한 output을 length 오름차순으로 출력해주기 위해서 L의 itemset마다 계산해주었다.

```
// ========================================
// ********Association Rule 구하는 함수********
// Output에 출력해줄 각 Association Rule과
// support, confidence 값을 구한다.
// ========================================
vector<information> find_Association_Rules(map_itemset_sup& all_L)
{
    vector<information> res;
    for (auto it = all_L.begin(); it != all_L.end(); it++)
    {
        set<int> Lk = it->first;
        int sup = it->second;

        if (Lk.size() < 2) // 1개 짜리 원소는 패스
            continue;

        set<set<int>> powersets = getPowerset(Lk);
        powersets.erase(Lk); // 자기 자신은 삭제

        for (auto& right_powerset : powersets)
        {
            set<int> left_powerset(Lk);
            set<int> result;
            set_difference(left_powerset.begin(), left_powerset.end(), right_powerset.begin(), right_powerset.end(), inserter(result, result.end()));

            res.push_back({ all_L, it, right_powerset, result });
        }
    }
    return res;
}
```

모든 Association rule을 구해주기 위해서는 하나의 아이템 셋에 대해서 모든 부분집합이 필요하다. 단, 자기 자신과 공집합은 제외한다.

```
struct information
{
    map_itemset_sup all_L;
    map_itemset_sup::iterator it;
    set<int> right_powerset;
    set<int> result;
};
```

추후에 write_Output에서 사용되기 좋게 따로 자료형을 선언하고 이를 원소로 하는 vector를 만들어서 push_back 해주었다.

```
// ============================================
// *************멱집합 구하는 함수*************
// Association Rule을 얻으려면 어떤 집합의
// 자기 자신과 공집합을 제외한 집합을 얻어야 함.
// 우선 멱집합을 구하고 후처리 한다.
// ============================================
candidates getPowerset(const set<int>& Lk)
{
    candidates powerset;

    for (auto& item : Lk)
    {
        // 새로 만든 부분 집합과 새로운 원소 사용해서 부분집합 생성
        for (auto& subset : powerset)
        {
            set<int> tmp(subset);
            tmp.insert(item);
            powerset.insert(tmp);
        }
        // 1개짜리 원소도 부분집합에 추가
        powerset.insert({ item });
    }
    return powerset;
}
```

getPowerset은 앞서 말한 집합들을 구하는 함수이다. 새로 만든 부분 집합과 새로운 원소를 사용해서 또 새로운 부분집합을 생성하고, 1 개짜리 원소는 따로 부분집합에 추가해준다. 이렇게 해서 구한 association rule들을 write_Output 함수에 넘겨준다.

```
// ========================================
// ****Output.txt에 결과물 출력해주는 함수****
// find_Association_Rules에서 얻은 정보로
// 출력 파일에 저장해준다.
// ========================================
void write_Output(vector<information>& v, ofstream& writeFile, string outputFileName)
{
    for (auto& idx : v)
    {
        string item_set, associative_item_set;
        item_set = associative_item_set = "{";
        for (auto& item : idx.right_powerset)
            item_set += (to_string(item) + ",");
        item_set.pop_back();

        for (auto& item : idx.result)
            associative_item_set += (to_string(item) + ",");
        associative_item_set.pop_back();

        double sup = (double)idx.it->second / DB.size() * 100;
        double confidence = (double)idx.it->second / all_frequent_set[idx.right_powerset] * 100;

        writeFile << item_set << "}₩t" << associative_item_set << "}₩t" << sup << '₩t' << confidence << '₩n';
    }
}
```

요구사항의 출력 제한에 맞추어서 갖고 있는 정보들을 넣어준다. Sup은 어차피 같이 저장해주었기 때문에 전체 DB의 size로 나눠주면 되고, confidence의 경우 조건부 확률이기 때문에, 아까 전체 L을 저장해준 all_frequent_set에서 찾아서 나눠준다.

4. Testing

강의 자료의 예제로 테스트 해본 결과 값이 알맞게 잘 나왔다.

```
ubuntu@ubuntu:~/Assignment1$ ./apriori.exe 40 input.txt output.txt
```

```
{1}     {3}     50.00   100.00
{3}     {1}     50.00   66.67
{2}     {3}     50.00   66.67
{3}     {2}     50.00   66.67
{2}     {5}     75.00   100.00
{5}     {2}     75.00   100.00
{3}     {5}     50.00   66.67
{5}     {3}     50.00   66.67
{2}     {3,5}   50.00   66.67
{2,3}   {5}     50.00   100.00
{2,5}   {3}     50.00   66.67
{3}     {2,5}   50.00   66.67
{3,5}   {2}     50.00   100.00
{5}     {2,3}   50.00   66.67
```