2014057983 컴퓨터공학과 지소현

1. Summary of Algorithm

First, we scan database so we can obtain support count and Candidate itemset C(k). We find frequent itemset L(k).L(k) is elements of Candidate itemset which is greater than minimum support.

Second, we join L(k) & L(k) and we can make Candidate itemset C(k+1).

Third, If there is any itemset which is infrequent, its superset should not be generated/tested(Pruning principle).

Finally, we can obtain association rule from L(k) and Remove the association rule which is not greater than minimum support. This operation repeats while there are no more new association rules.

2.Detailed description of codes

#include <iostream>

#include <stdio.h>

#include <fstream>

#include <vector>

#include <set>

#include <math.h>

#include <string>

#include <sstream>

#define ROUNDING(x, dig) ( floor((x) \* pow(float(10), dig) + 0.5f) / pow(float(10), dig) )

using namespace std;

int minSupport;

int size\_transaction;

char\* output\_name;

//struct for saving frequent itemset, candidate itemset and each itemset’s support\_count

struct Itemset{

set<int> frequent\_itemset;

int support\_count;

//constructor Itemset

Itemset(set<int> itemset\_,int suppport\_count\_) {

frequent\_itemset.insert(itemset\_.begin(), itemset\_.end());

support\_count = suppport\_count\_;

}

};

//this function checks that array has number.

int IsSubset(int num,vector<int> \*array){

vector<int>::iterator it;

it = find (array->begin(), array->end(), num );

if (it != array->end()) {

return 1;

}

return 0;

}

//transaction is input text file

//and set<int> is array which is wanted to know(get) support

int getSupport(vector<vector<int> > transaction,set<int> \*tempCset) {

//temperate set save in ‘temp’ array.

int \*temp = new int[tempCset->size()];

int current=0,support=0;

for(set<int>::iterator it = tempCset->begin(); it != tempCset->end(); it++) {

temp[current++] = (\*it);

}

//save transacntion matrix’s row unit

vector<int> transaction\_row;

for(int i=0; i<transaction.size(); i++) {

transaction\_row = transaction.at(i);

int cnt=0,res=1;

for(int i=0; i<current; i++) {

//find that temp[i] is in transaction’s row

//if find it, found element’s count +1

//else, break.

res = IsSubset(temp[i], &transaction\_row);

if(res==0) {

break;

} else {

cnt++;

}

}

//if find, res=1.

//else, res=0 and support count +1.

if(res==1){

support++;

}

transaction\_row.clear();

}

//return found support count

return support;

}

//join C(k) and C(k), So make C(k+1)

//ex) if k=1…and we make Candidate Set(3).

int make\_Candidate\_sets(vector<vector<int> > transaction,vector<vector<Itemset> > \*L,int k) {

set<int> cset;

vector<Itemset> Ck; // C(k) : candidate itemset of k+1 th

int count = 0;

//before insert joined itemset in C(k)’s transation, check duplication.

for(int i=0; i<L->at(k).size(); i++) {

for(int j=i+1; j<L->at(k).size(); j++) {

cset.insert( L->at(k).at(i).frequent\_itemset.begin(),

L->at(k).at(i).frequent\_itemset.end()) ;

cset.insert( L->at(k).at(j).frequent\_itemset.begin(),

L->at(k).at(j).frequent\_itemset.end()) ;

//set hasn’t duplicated elements.

//So, if its size is original size+1, joining is succeed

if(cset.size()== (k+2)) {

int ISsame = -1;

//first itemset must insert because first itemset is always not duplicate(we can know first itemset..because C(k)’s count is 0.)

if ( count==0 ) {

//itemset is more than min\_support, insert it.

if(getSupport(transaction,&cset)>=minSupport) {

Itemset itemset(cset,getSupport(transaction,&cset));

Ck.push\_back(itemset);

count ++;

}

} else { //if it is not first itemset, we must check itemset’s duplication.

for(int i=0; i<Ck.size(); i++) {

if(cset==Ck.at(i).frequent\_itemset) {

ISsame = 1;

}

}

//if itemset is not duplicate, insert itemset which is greater than min\_support in C(K).

if(ISsame!=1) {

if(getSupport(transaction,&cset)>=minSupport) {

Itemset itemset(cset,getSupport(transaction,&cset));

Ck.push\_back(itemset);

count++;

}

}

}

}

cset.clear();

}

}

//completed C(K) insert in L.

L->push\_back(Ck);

Ck.clear();

return count;

}

// When it is association rule (A=>B), outputfile create.

void print(vector<vector<int> > transaction,int result[], int length,int \*arr,int arr\_num) {

ofstream output(output\_name,ios::app);

string s;

stringstream ss; //for output file’s buffer

set<int> A ,B,AnB;

int current =0,isSame;

int totalSupport;

double Support, Confidence;

if(length>0) {

// print {A}

ss << "{" ;

for (int i = 0; i < length; i++){

ss << result[i] ;

A.insert(result[i]);

if(i != length-1) {

ss << "," ;

}

}

ss << "}" ;

//print {B}.. insert elements which are not included in A.

for(int j=0; j<arr\_num; j++) {

isSame=0;

for (set<int>::iterator it=A.begin(); it!=A.end(); it++) {

if(arr[j] == (\*it)){

isSame =1; //if number is same with A’s element,change 1..in order not to insert it in B.

}

}

//else number is not same, insert it in B.

if(isSame!=1) {

B.insert(arr[j]);

}

}

ss << "\t" << "{" ;

for(set<int>::iterator it=B.begin(); it !=B.end(); it++) {

ss << (\*it) ; current++;

if(current < B.size()) {

ss << "," ;

}

}

ss << "}" << "\t" ;

//AnB is itemset which include all A and B’s elements to calculate Total support.

AnB.insert( A.begin(),A.end() ) ;

AnB.insert( B.begin(),B.end() ) ;

totalSupport = getSupport(transaction,&AnB);

Support = (totalSupport / (double)size\_transaction) \*100;

Confidence = (totalSupport / (double)getSupport(transaction,&A)) \*100;

//for denote decimal represents up to two digits.

ss.precision(2);

ss <<fixed ;

ss << ROUNDING(Support, 2) << "\t" << ROUNDING(Confidence, 2) << endl;

}

s= ss.str();

output << s ;

}

// create a subset without duplicates in the ‘arr’array using combination.

void Combination(vector<vector<int> > transaction,int \*result,int index, int n, int r, int target,int \*arr) {

if (r == 0) {

print(transaction,result, index,arr,n);

} else if (target == n) {

return;

} else {

result[index] = arr[target];

Combination(transaction, result, index+1, n, r-1, target+1, arr);

Combination(transaction, result, index, n, r, target+1, arr);

}

}

// obtaining association rules about L(k+1)

int DivideSubset(vector<vector<int> > transaction,vector<vector<Itemset> > \*L,int k) {

// temporarily save L(i)’s itemset for create a subset without duplicates.

int \*temp,q;

int \*result;

for(int i=0; i<L->at(k).size(); i++) {

int size = (int)L->at(k).at(i).frequent\_itemset.size();

temp= new int[size]; q=0;

result = new int[size];

for(set<int>::iterator it = L->at(k).at(i).frequent\_itemset.begin() ;

it != L->at(k).at(i).frequent\_itemset.end(); it++) {

temp[q++] = (\*it);

}

for(int i=0; i<size; i++) {

Combination(transaction, result,0,size,i, 0,temp);

}

}

return 0;

}

int main(int argc, char \*\* argv) {

int relative\_support = atoi(argv[1]);

char\* input\_name = argv[2];

output\_name = argv[3];

char str[50];

ifstream input;

input.open(input\_name);

//read input text and save total transaction

vector<vector<int> > transaction;

vector<int> array;

set<int> candidate\_set;

size\_transaction=0;

if(input.is\_open()) {

while(input.getline(str, sizeof(str))) {

size\_transaction++;

char \*ptr = strtok(str, "\t"); // truncate strings based on “ “whitespace,and return a pointer

while (ptr != NULL) //repeat until truncated string

{

int number = atoi(ptr);

array.push\_back(number);

candidate\_set.insert(number);

ptr = strtok(NULL, "\t"); //cut the next string and return a pointer

}

transaction.push\_back(array);

array.clear();

}

}

//obtain absolute support using relative support

minSupport = ceil(((double)size\_transaction / 100) \* relative\_support);

int support[candidate\_set.size()];

memset(support,0,candidate\_set.size()\*sizeof(int));

for(int i=0; i<transaction.size(); i++) {

for(int j=0; j<transaction[i].size(); j++) {

support[transaction[i][j]]++;

}

}

//find L1.

vector<vector<Itemset> > L; // L(i) : frequent itemset of size i

vector<Itemset> Lk;

set<int> Lk\_temp;

int \*my\_set = new int[candidate\_set.size()],q=0;

for(int i=0; i<candidate\_set.size(); i++) {

if(minSupport<=support[i]){

Lk\_temp.insert(i);

my\_set[q++] = i;

Itemset itemset(Lk\_temp,support[i]);

Lk.push\_back(itemset);

Lk\_temp.clear();

}

}

L.push\_back(Lk);

Lk\_temp.clear(); Lk.clear();

//find L2

for( int first = 0; first < q; ++first ) {

for( int second = first+1; second < q; ++second ) {

if( first != second ) {

Lk\_temp.insert(my\_set[first]);

Lk\_temp.insert(my\_set[second]);

if(getSupport(transaction,&Lk\_temp)>=minSupport) {

Itemset itemset(Lk\_temp,getSupport(transaction,&Lk\_temp));

Lk.push\_back(itemset);

}

Lk\_temp.clear();

}

}

}

L.push\_back(Lk);

Lk\_temp.clear();

Lk.clear();

//Find C(K) and insert itemset which is greater than min support in L(k)

int res=9999,i=1;

while(1) {

res = make\_Candidate\_sets(transaction,&L, i);

if(res==1 || res==0) {

break;

}

i++;

}

input.close();

ofstream output(output\_name,ios\_base::out | ios\_base::binary | ios\_base::trunc);

//find L(k)’s subset and association rule…and print result.

for(int i=0; i< L.size(); i++) {

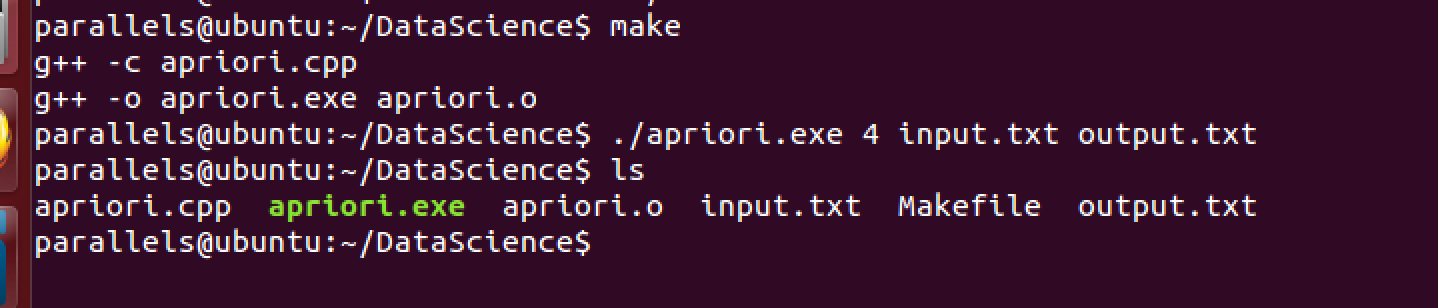
DivideSubset(transaction,&L, i);

}

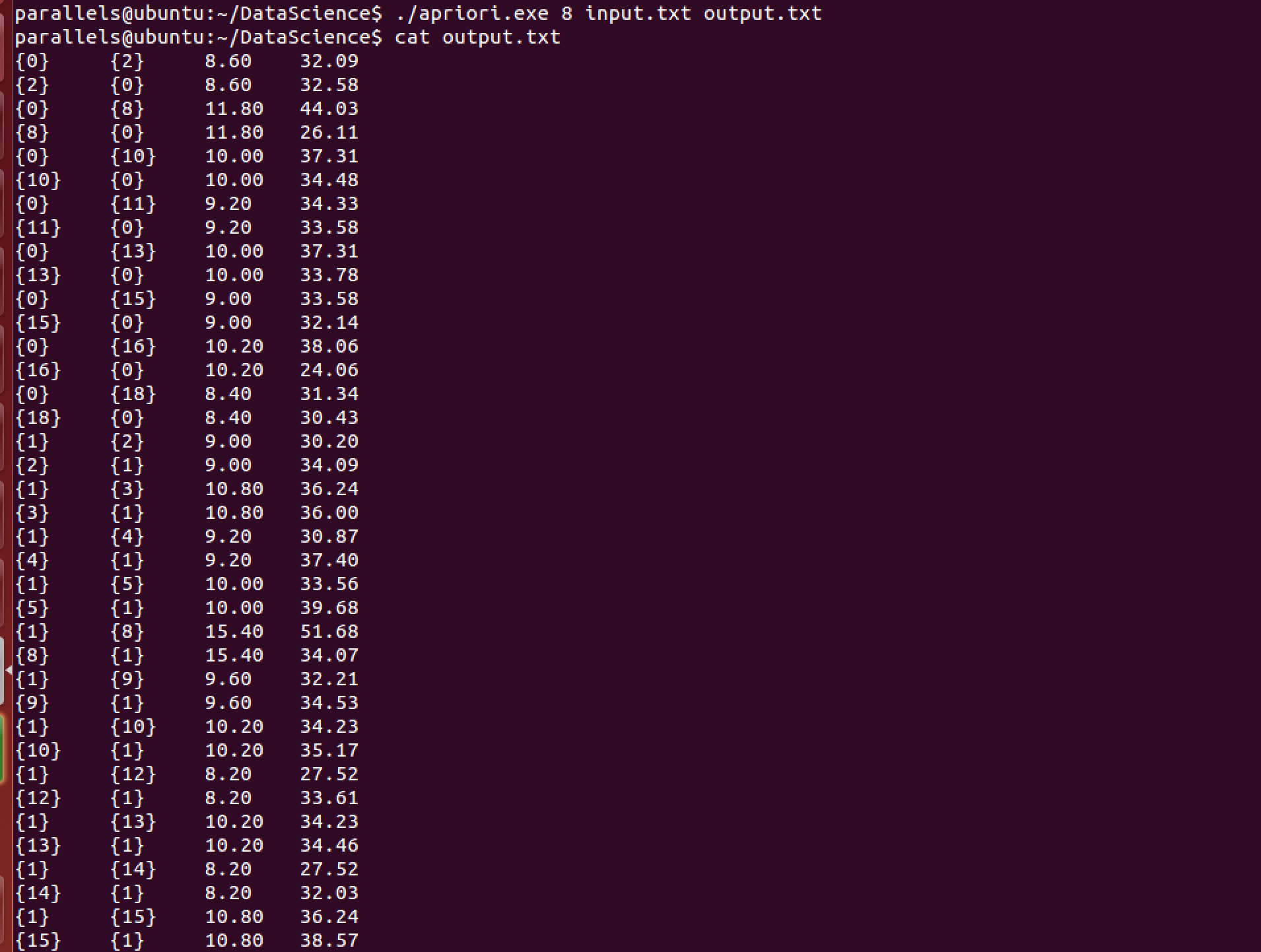
return 0;

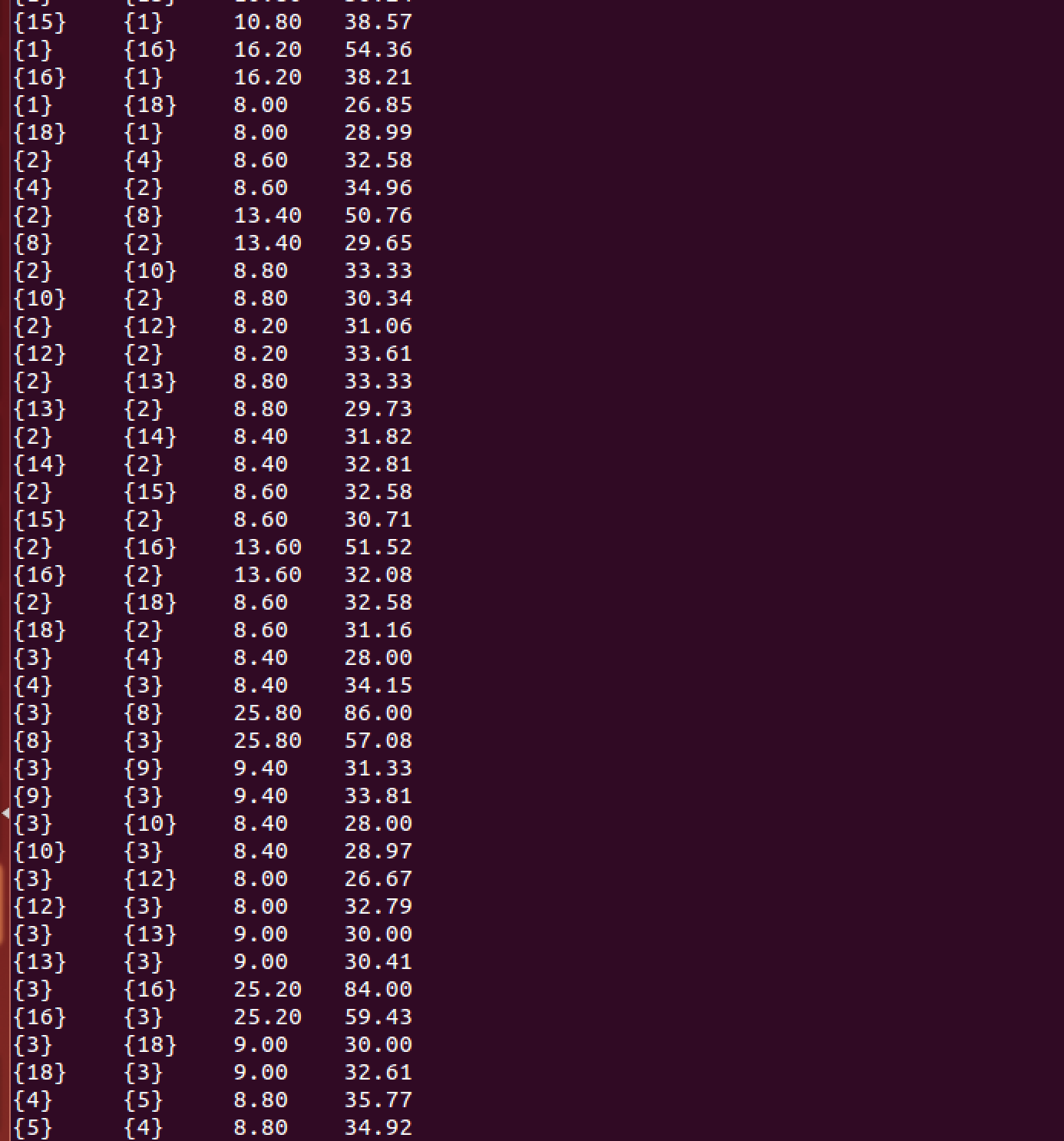
}

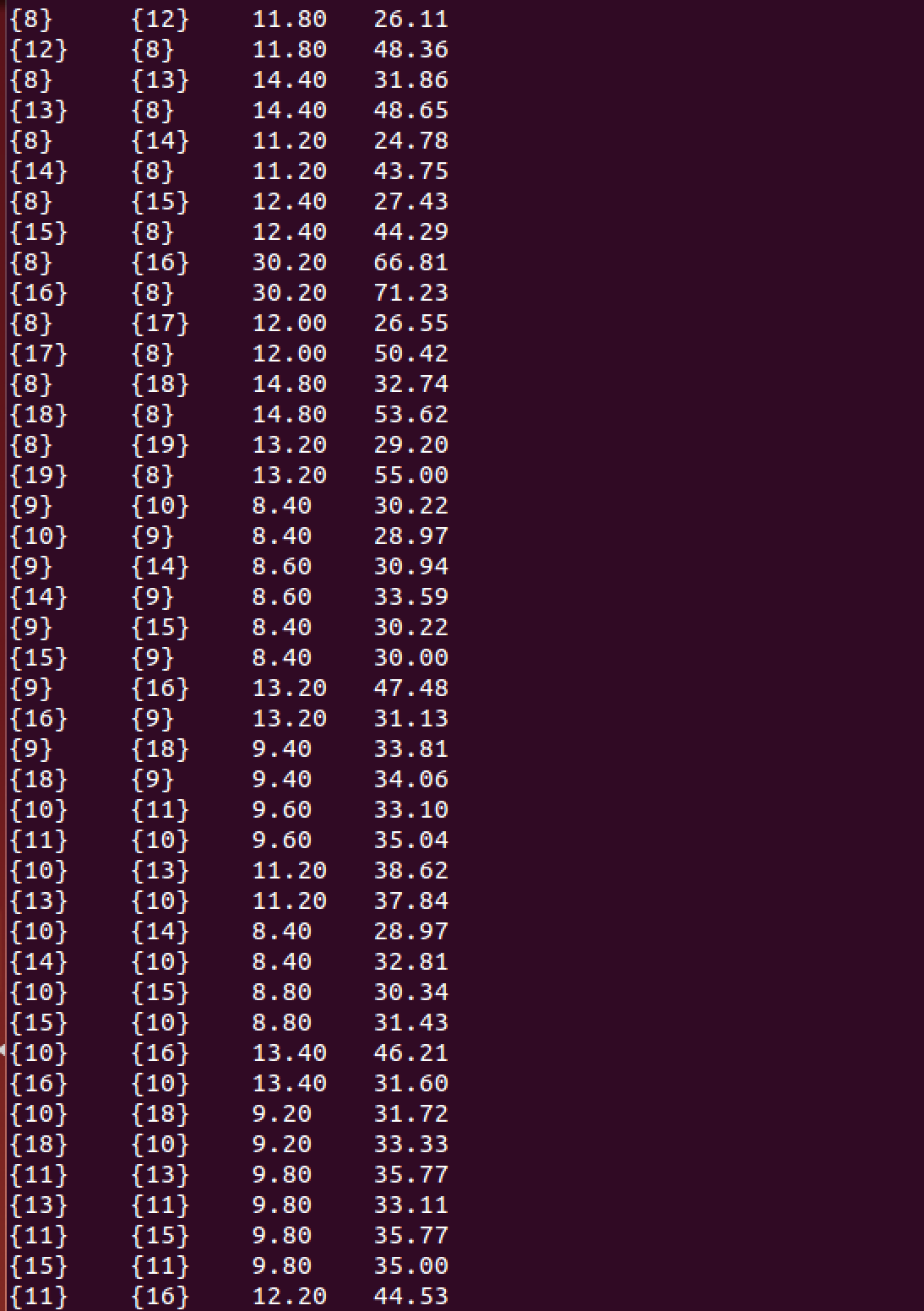
1. Insrtuctions for compilling my source code(e.g screenshot)

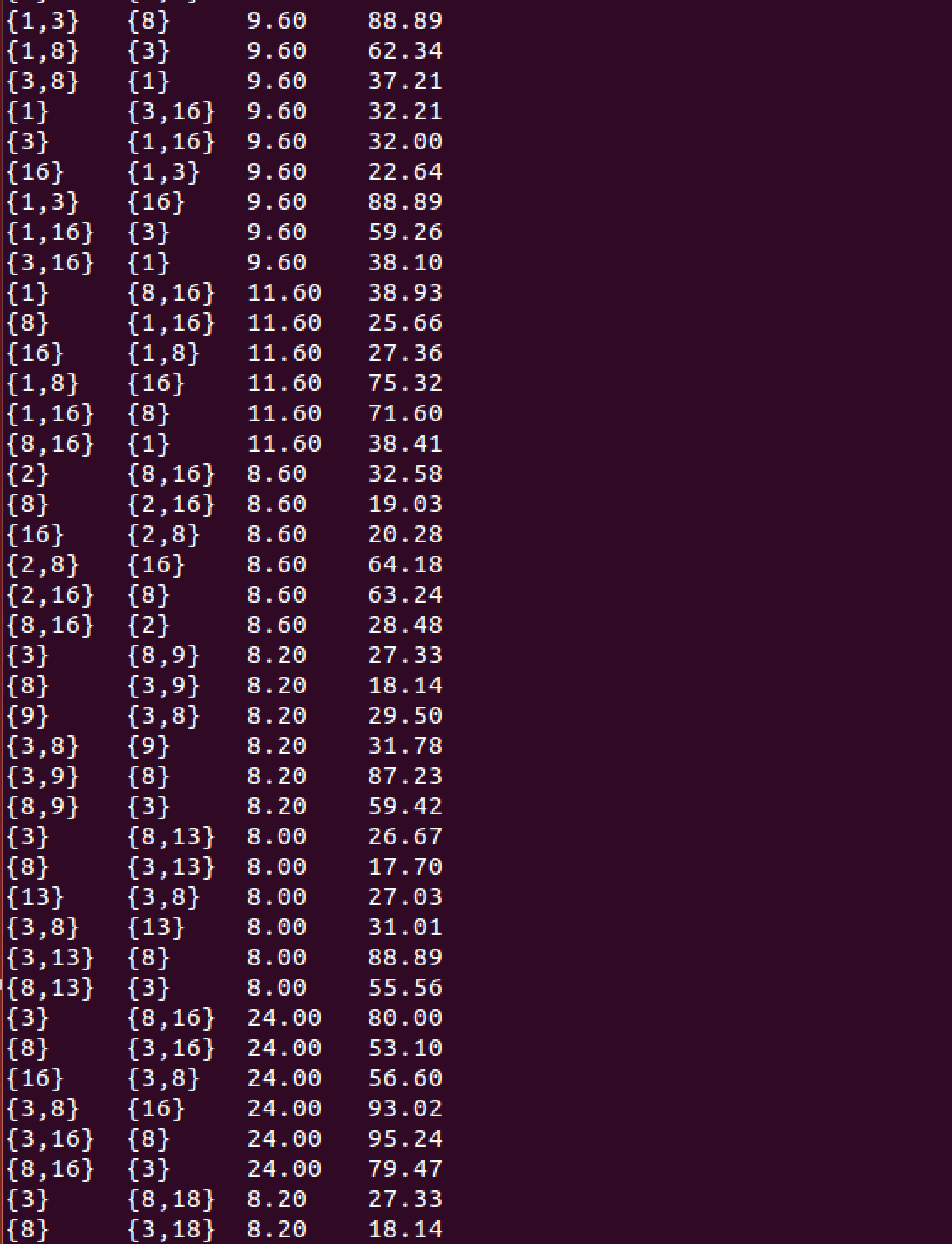
make -> ./apriori.exe support inputfileNAME outputfileNAME  


1. Any other specification of my implementation and testing







…and Middle content omitted.

