Experiment no. 8

Name: Sonali Dattatray Kaingade

PRN: 21620002

Title: Extend program 7, to find association rule.

code:

```
#include <bits/stdc++.h>

#include <map>

using namespace std;


ifstream fin;

double minfre;

vector<set<string>> datatable;

set<string> products;

map<string, int> freq;

double confidence;


// Function to split a string into words

vector<string> wordsof(string str)

{

   vector<string> tmpset;

   string tmp = "";

   int i = 0;

   while (str[i])

   {

      if (isalnum(str[i]))
```

```
            tmp += str[i];
        else
        {
            if (tmp.size() > 0)
                tmpset.push_back(tmp);
            tmp = "";
        }
        i++;
    }

    if (tmp.size() > 0)
        tmpset.push_back(tmp);

    return tmpset;
}

// Function to combine a vector of strings, excluding the one at a given index
string combine(vector<string> &arr, int miss)
{
    string str;
    for (int i = 0; i < arr.size(); i++)
        if (i != miss)
            str += arr[i] + " ";
    str = str.substr(0, str.size() - 1);
    return str;
```

```
}

// Function to clone a set of strings
set<string> cloneit(set<string> &arr)
{
    set<string> dup;
    for (set<string>::iterator it = arr.begin(); it != arr.end(); it++)
        dup.insert(*it);
    return dup;
}

// Generate candidate itemsets for Apriori
set<string> apriori_gen(set<string> &sets, int k)
{
    set<string> set2;
    for (set<string>::iterator it1 = sets.begin(); it1 != sets.end(); it1++)
    {
        set<string>::iterator it2 = it1;
        it2++;
        for (; it2 != sets.end(); it2++)
        {
            vector<string> v1 = wordsof(*it1);
            vector<string> v2 = wordsof(*it2);

            bool alleq = true;
            for (int i = 0; i < k - 1 && alleq; i++)
                if (v1[i] != v2[i])
                    alleq = false;

            v1.push_back(v2[k - 1]);
            if (v1[v1.size() - 1] < v1[v1.size() - 2])
                swap(v1[v1.size() - 1], v1[v1.size() - 2]);

            for (int i = 0; i < v1.size() && alleq; i++)
            {
                string tmp = combine(v1, i);
                if (sets.find(tmp) == sets.end())
                    alleq = false;
            }
            if (alleq)
                set2.insert(combine(v1, -1));
        }
    }
    return set2;
}

// Count occurrences of a set of items in the dataset
int countOccurences(vector<string> v)
{
    int count = 0;
    for (auto s : datatable)
    {
        bool present = true;

        for (auto x : v)
        {
            if (s.find(x) == s.end())
            {
                present = false;
                break;
            }
        }

        if (present)
            count++;
    }

    return count;
}

ofstream fw1("association_output.csv", ios::out);

// Generate subsets of items for association rule generation
void subsets(vector<string> items, vector<string> v1, vector<string> v2, int idx)
```

```
{
    if (idx == items.size())
    {
        if (v1.size() == 0 || v2.size() == 0)
            return;

        int count1 = countOccurences(items); // Total support
        int count2 = countOccurences(v1);

        double conf = (((double)count1) / count2) * 100;

        if (conf >= confidence)
        {
            fw1 << "Association Rule: { ";
            for (auto s : v1)
            {
                fw1 << s << " ";
            }
            fw1 << "} -> {";
            for (auto s : v2)
            {
                fw1 << s << " ";
            }
            fw1 << "} , Confidence: " << conf << "%" << endl;
        }
        return;
    }


    v1.push_back(items[idx]);
    subsets(items, v1, v2, idx + 1);


    v1.pop_back();
    v2.push_back(items[idx]);
    subsets(items, v1, v2, idx + 1);
    v2.pop_back();
}


// Generate association rules from frequent itemsets
void generateAssociationRules(set<string> freqItems)
{
    for (auto it = freqItems.begin(); it != freqItems.end(); it++)
    {
        vector<string> items = wordsof(*it);


        subsets(items, {}, {}, 0);
    }
}


int main()
```

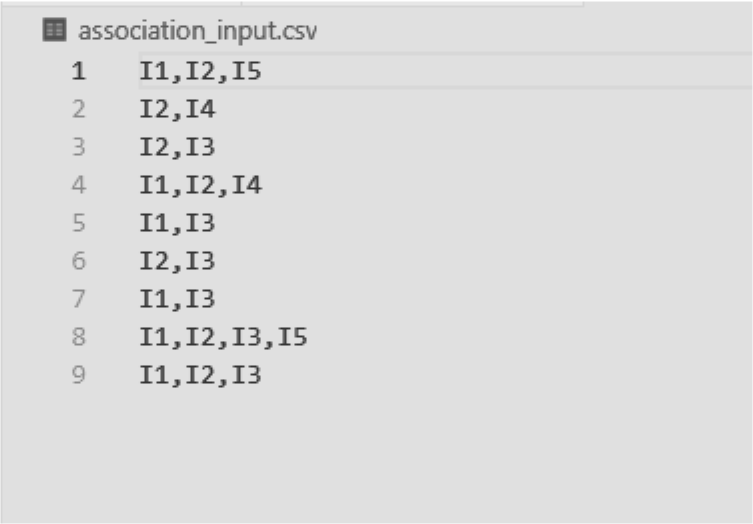
```
{
    fin.open("association_input.csv", ios::in);

    if (!fin.is_open())
    {
        cerr << "Error in opening file." << endl;
        return 1;
    }

    cout << "Enter Minimum Support (%): ";
    cin >> minfre;

    cout << "Enter Minimum Confidence (%): ";
    cin >> confidence;

    string str;
    while (!fin.eof())
    {
        getline(fin, str);
        vector<string> arr = wordsof(str);
        set<string> tmpset;
        for (int i = 0; i < arr.size(); i++)
            tmpset.insert(arr[i]);
        datatable.push_back(tmpset);
    for (set<string>::iterator it = tmpset.begin(); it != tmpset.end(); it++)
    {
        products.insert(*it);
        freq[*it]++;
    }
}
fin.close();

cout << "Number of transactions: " << datatable.size() << endl;
minfre = minfre * datatable.size() / 100;
cout << "Minimum Frequency Threshold: " << minfre << endl;

queue<set<string>::iterator> q;
for (set<string>::iterator it = products.begin(); it != products.end(); it++)
    if (freq[*it] < minfre)
        q.push(it);

while (q.size() > 0)
{
    products.erase(*q.front());
    q.pop();
}

int pass = 1;
cout << "Frequent " << pass++ << "-item set: " << endl;
```

```
for (set<string>::iterator it = products.begin(); it != products.end(); it++)
    cout << "{" << *it << "} - Support: " << freq[*it] << endl;

int i = 2;
set<string> prev = cloneit(products);

while (i)
{
    set<string> cur = apriori_gen(prev, i - 1);

    if (cur.size() < 1)
    {
        break;
    }

    for (set<string>::iterator it = cur.begin(); it != cur.end(); it++)
    {
        vector<string> arr = wordsof(*it);

        int tot = 0;
        for (int j = 0; j < datatable.size(); j++)
        {
            bool pres = true;
            for (int k = 0; k < arr.size() && pres; k++)
                if (datatable[j].find(arr[k]) == datatable[j].end())
                pres = false;
        if (pres)
            tot++;
    }
    if (tot >= minfre)
        freq[*it] += tot;
    else
        q.push(it);
}

while (q.size() > 0)
{
    cur.erase(*q.front());
    q.pop();
}

bool flag = true;

for (set<string>::iterator it = cur.begin(); it != cur.end(); it++)
{
    vector<string> arr = wordsof(*it);

    if (freq[*it] < minfre)
        flag = false;
}
```

```
        if (cur.size() == 0)
            break;

        cout << "\nFrequent " << pass++ << "-item set: " << endl;
        for (set<string>::iterator it = cur.begin(); it != cur.end(); it++)
            cout << "{" << *it << "} - Support: " << freq[*it] << endl;

        prev = cloneit(cur);
        i++;
    }

    generateAssociationRules(prev);

    cout << "Association rules generated successfully." << endl;

    return 0;
}
```

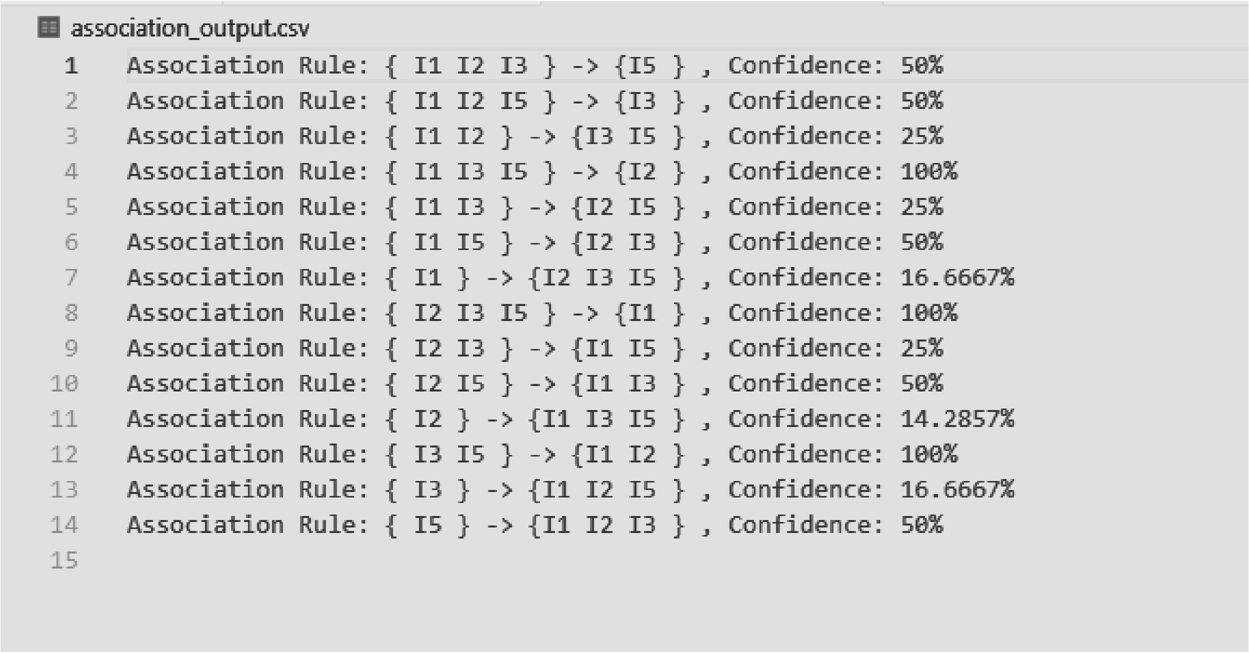
Output:

input.csv:

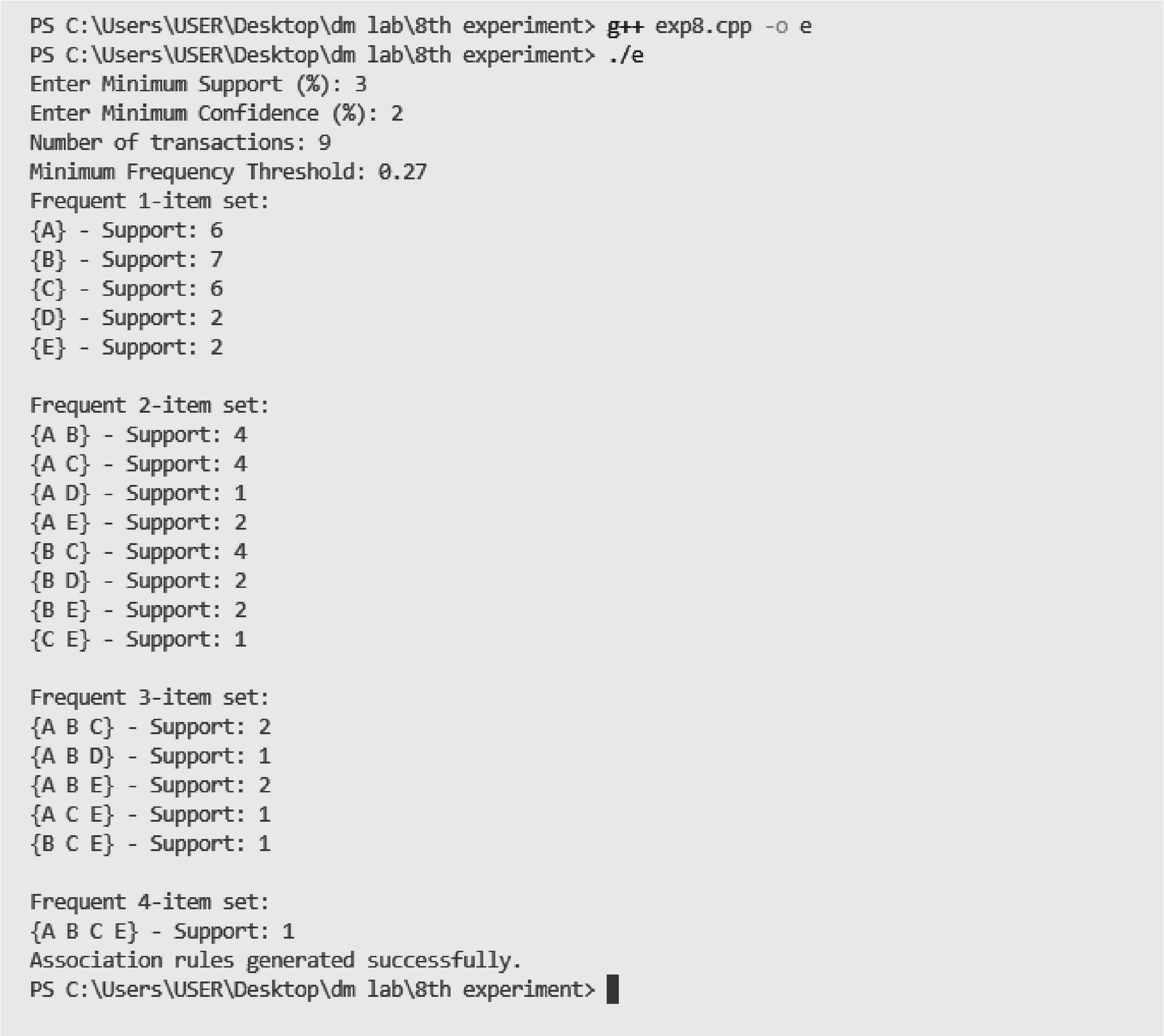


```
I1,I2,I5
I2,I4
I2,I3
I1,I2,I4
I1,I3
I2,I3
I1,I3
I1,I2,I3,I5
I1,I2,I3
```

output.csv:

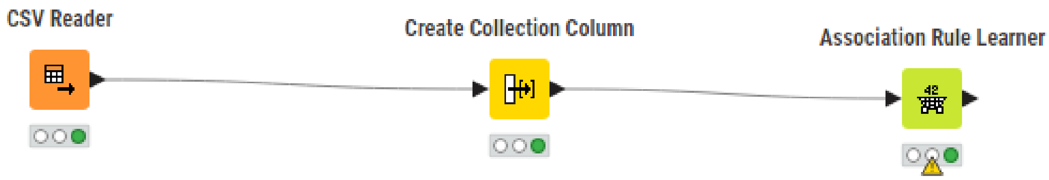


```
Association Rule: { I1 I2 I3 } -> {I5 } , Confidence: 50%
Association Rule: { I1 I2 I5 } -> {I3 } , Confidence: 50%
Association Rule: { I1 I2 } -> {I3 I5 } , Confidence: 25%
Association Rule: { I1 I3 I5 } -> {I2 } , Confidence: 100%
Association Rule: { I1 I3 } -> {I2 I5 } , Confidence: 25%
Association Rule: { I1 I5 } -> {I2 I3 } , Confidence: 50%
Association Rule: { I1 } -> {I2 I3 I5 } , Confidence: 16.6667%
Association Rule: { I2 I3 I5 } -> {I1 } , Confidence: 100%
Association Rule: { I2 I3 } -> {I1 I5 } , Confidence: 25%
Association Rule: { I2 I5 } -> {I1 I3 } , Confidence: 50%
Association Rule: { I2 } -> {I1 I3 I5 } , Confidence: 14.2857%
Association Rule: { I3 I5 } -> {I1 I2 } , Confidence: 100%
Association Rule: { I3 } -> {I1 I2 I5 } , Confidence: 16.6667%
Association Rule: { I5 } -> {I1 I2 I3 } , Confidence: 50%
```



```
PS C:\Users\USER\Desktop\dm lab\8th experiment> g++ exp8.cpp -o e
PS C:\Users\USER\Desktop\dm lab\8th experiment> ./e
Enter Minimum Support (%): 3
Enter Minimum Confidence (%): 2
Number of transactions: 9
Minimum Frequency Threshold: 0.27
Frequent 1-item set:
{A} - Support: 6
{B} - Support: 7
{C} - Support: 6
{D} - Support: 2
{E} - Support: 2

Frequent 2-item set:
{A B} - Support: 4
{A C} - Support: 4
{A D} - Support: 1
{A E} - Support: 2
{B C} - Support: 4
{B D} - Support: 2
{B E} - Support: 2
{C E} - Support: 1

Frequent 3-item set:
{A B C} - Support: 2
{A B D} - Support: 1
{A B E} - Support: 2
{A C E} - Support: 1
{B C E} - Support: 1

Frequent 4-item set:
{A B C E} - Support: 1
Association rules generated successfully.
PS C:\Users\USER\Desktop\dm lab\8th experiment> █
```

knime:


CSV Reader
Create Collection Column
Association Rule Learner