Experiment no. 7

Name: Sonali Dattatray Kaingade

PRN: 21620002

Title: Find frequent itemset from given transaction data.

Code:

```
#include <bits/stdc++.h>
#include <map>
using namespace std;

double minfre;              // Minimum frequency for itemsets to be considered frequent.
vector<set<string>> datatable; // A vector to store transaction data.
set<string> products;       // Set to store unique products/items in the transactions.
map<string, int> freq;      // A map to store the frequency of each product/item.

// Function to split a string into words based on alphanumeric characters.
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 elements in a vector into a string, excluding the one at 'miss' 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 and return a copy.
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;
}

// Function to generate frequent itemsets of size k based on candidate itemsets of size k-1.
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);
         // mergig v1 and v2 like lattice i.e permutations and combinations type
         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;
}


int main()
{
    ifstream fin("item_set_input.csv", ios::in); // Open the input file for reading.


    if (!fin.is_open())
```

```
{
    perror("Error in opening file : "); // Print an error message if the file cannot be opened.
}
cout << "Enter min Frequency  :";
cin >> minfre; // Read the minimum frequency from the user.


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); // Store the transaction data in the 'datatable' vector.


    for (set<string>::iterator it = tmpset.begin(); it != tmpset.end(); it++)
    {
        products.insert(*it); // Store unique products in the 'products' set.
        freq[*it]++;         // Increment the frequency of each product in the 'freq' map.
    }
}
fin.close(); // Close the input file.
```

```
cout << "No of transactions: " << datatable.size() << endl;
// minfre = minfre * datatable.size() / 100; // Calculate the minimum frequency threshold.
cout << "Min frequency:" << 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()); // Remove infrequent products from the 'products' set.
   q.pop();
}

int pass = 1;
cout << "\nFrequent " << pass++ << " -item set : \n";
for (set<string>::iterator it = products.begin(); it != products.end(); it++)
   cout << "{" << *it << "} " << freq[*it] << endl; // Display frequent 1-itemsets.

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

while (i)
```

```
{
    set<string> cur = apriori_gen(prev, i - 1); // Generate candidate itemsets of size 'i'.

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

```
        prev = cloneit(cur);
        i++;
    }
    ofstream fw("item_set_output.csv", ios::out); // Open an output file for writing.

    for (auto it = prev.begin(); it != prev.end(); it++)
    {
        fw << "{" << *it << "}" << endl; // Write frequent itemsets to the output file.
    }

    return 1;
}
```

Result:

Input:

	A	B	C	D
1	A	B	C	
2	A	D		
3	B	C		
4	A	B	D	
5	A	B	C	D
6				
7				
8				

Output:

```
PS E:\second year engg IT\7th sem\dm lab\7th experiment> g++ frequentitem.cpp -o freit
PS E:\second year engg IT\7th sem\dm lab\7th experiment> ./freit
Frequency % :60
No of transactions: 5
Min frequency:3

Frequent 1 -item set :
{A} 4
{B} 4
{C} 3
{D} 3


Frequent 2 -item set :
{A B} 3
{A D} 3
{B C} 3
PS E:\second year engg IT\7th sem\dm lab\7th experiment>
```

Frequent-itemsets are:

	A	B
1	{A B}	
2	{A D}	
3	{B C}	
4		
5		