1. Write a program to implement a data cube to perform operations on it and store the implementations in a file.

#include <iostream>

#include<fstream>

using namespace std;

int main()

{

ofstream outdata;

outdata.open("data\_cube\_sample.csv");

int arr[3][3][3], i, j, k, a, b, c;

int entered\_S\_No,entered\_semester,choosen\_attribute;

const char\* column\_val[10] = {"S.No", "Age","Sleeping habbit"};

cout<<"Enter the dimensions";

cout<<"\*\*\*\*\*"<<endl;

cout<<"Enter the size of the semester dimensions\n";

cin>>a;

cout<<"Enter the size of the student dimensions\n";

cin>>b;

cout<<"Enter the size of the attribute dimensions\n";

cin>>c;

for(i=0;i<a;i++)

{

for(j=0;j<b;j++)

{

for(k=0;k<c;k++)

{

cout<<"Enter the "<<column\_val[k]<<" of student: "<<j+1<<"in sem: "<<i+1<<endl;

cout<<"location\t"<<i<<"\t"<<j<<"\t"<<k<<endl;

cin>>arr[i][j][k];

}

}

}

cout<<"value extractions from data cube \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Enter the student Serial Number: "<<endl;

cin>>entered\_S\_No;

cout<<"Enter the semester"<<endl;

cin>>entered\_semester;

cout<<"Enter the attribute you want to fetch"<<endl;

cout<<"1. S.No \n2.Age \n3.Sleeping habbit"<<endl;

cin>>choosen\_attribute;

outdata<<"required attribute"<<endl;

if(choosen\_attribute<1)

{

cout<<"Enter a valid attribute"<<endl;

}

else

{

cout<<"\n"<<endl;

cout<<arr[entered\_semester-1][entered\_S\_No-1][choosen\_attribute-1]<<endl;

//outdata<<"\n"<<endl;

outdata<<arr[entered\_semester-1][entered\_S\_No-1][choosen\_attribute-1]<<endl;

}

return 0;

}

2. Write a program to implement a data cube for the following Tourist's data for 5 years in 3 states quarter wise and perform either a SLICE or DICE operations on the data cube and save the operations in a file.

#include<iostream>

#include<fstream>

#include<cstdio>

#include<cstdlib>

using namespace std;

int main()

{

ofstream outdata;

outdata.open("sample.csv");

int arr[5][5][5],i,j,k,a,b,c,option=0,state,quarter,att,year;

int enter\_S\_no,enter\_semsester,chosen\_attribute;

const char\* years[10]={"2017","2018","2019","2020","2021"};

const char\* states[10]={"Assam","Kerela","Goa"};

const char\* column\_val[10]={"q1","q2","q3","q4"};

cout<<"Tourists in States"<<endl;

cout<<"enter the dimensions"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"enter the size of the year dimensions"<<endl;

cin>>a;

cout<<"enter the size of the state dimensions"<<endl;

cin>>b;

cout<<"enter the size of the quarter attributes"<<endl;

cin>>c;

for(i=0;i<a;i++)

{

for(j=0;j<b;j++)

{

for(k=0;k<c;k++)

{

cout<<"enter the quarter attributes "<<column\_val[k]<<" of state: "<<states[j]<<" in year: "<<years[i]<<endl;

cout<<"location\t"<<i<<"\t"<<j<<"\t"<<k<<endl;

cin>>arr[i][j][k];

}

}

}

outdata<<"Data Representation"<<endl;

for(i=0;i<a;i++)

{

for(j=0;j<b;j++)

{

for(k=0;k<c;k++)

{

cout<<arr[i][j][k]<<"\t";

outdata<<arr[i][j][k]<<"\t";

}

cout<<"\t";

outdata<<"\t";

}

cout<<"\n";

outdata<<"\n";

}outdata<<"\n";

do

{

cout<<"Enter the dimension along which you want to slice"<<endl;

cout<<"1. year wise"<<endl;

cout<<"2. state Row wise"<<endl;

cout<<"3. quarter Attribute wise"<<endl;

cout<<"4. Exit"<<endl;

cin>>option;

switch(option)

{

case 1:

{

cout<<"Enter the year you want to extract"<<endl;

for(i=0;i<a;i++)

{

cout<<"year - "<<i+1<<"-"<<years[i]<<endl;

}

cin>>year;

outdata<<"years"<<endl;

for(j=0;j<b;j++)

{

for(k=0;k<c;k++)

{

cout<<arr[year-1][j][k]<<"\t";

outdata<<arr[year-1][j][k]<<"\t";

}outdata<<"\t";

}outdata<<"\n";

break;

}

case 2:

{

cout<<"Enter the state number you want to extract"<<endl;

for(j=0;j<b;j++)

{

cout<<"state - "<<j+1<<"-"<<states[j]<<endl;

}

cin>>state;

outdata<<"states"<<endl;

for(i=0;i<a;i++)

{

for(k=0;k<c;k++)

{

cout<<arr[i][state-1][k]<<"\t";

outdata<<arr[i][state-1][k]<<"\t";

}outdata<<"\t";

}outdata<<"\n";

break;

}

case 3:

{

cout<<"Enter the quarter Attribute you want to slice"<<endl;

for(k=0;k<c;k++)

{

cout<<column\_val[k]<<"-"<<k+1<<endl;

}

cin>>att;

outdata<<"quarter attributes"<<endl;

for(i=0;i<a;i++)

{

for(j=0;j<b;j++)

{

cout<<arr[i][j][att-1]<<"\t";

outdata<<arr[i][j][att-1]<<"\t";

}outdata<<"\t";

}outdata<<"\n";

outdata.close();

break;

}

case 4:

{

exit(3);

break;

}

}

}while(option!=4);

return 0;

}

Output:

3.Write a program in python to find the power set of a given set.

def power\_set(input\_set):

# Initialize the result with an empty set

result = [[]]

# Loop over the input set

for elem in input\_set:

# Loop over the subsets generated so far

for subset in result[:]:

# Add the current element to the subset and append to the result

result.append(subset + [elem])

return result

# Get the input set from the user

input\_set = set(input("Enter the set elements separated by spaces: ").split())

# Get the power set of the input set

output\_set = power\_set(input\_set)

# Print the power set

print("Power set of", input\_set, "is:")

for subset in output\_set:

print(subset)

Output:

4. Write a python program to implement the candidate generation algorithm.

def candidate\_generation(input\_set):

# Initialize an empty list to store the subsets

subsets = []

# Iterate through all possible subsets

for i in range(2\*\*len(input\_set)):

subset = []

# Iterate through all elements in the input set

for j in range(len(input\_set)):

# Check if the j-th bit in i is set

if i & (1 << j):

subset.append(input\_set[j])

subsets.append(subset)

return subsets

input\_set=str(input())

subsets = candidate\_generation(input\_set)

print(subsets)

Output:

5. Write a program in python to implement the Apriori algorithm.

import pandas as pd

from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules

df = pd.read\_csv('transactions.csv')

transactions = [x.strip().split(',') for x in df['Items Purchased'].tolist()]

items = list(set([item for transaction in transactions for item in transaction]))

transaction\_data = []

for transaction in transactions:

row = []

for item in items:

if item in transaction:

row.append(1)

else:

row.append(0)

transaction\_data.append(row)

transactions\_df = pd.DataFrame(transaction\_data, columns=items)

frequent\_itemsets = apriori(transactions\_df, min\_support=0.3, use\_colnames=True)

# Generate association rules from the frequent itemsets

association\_rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.15)

print("Frequent Itemsets:")

print(frequent\_itemsets)

print("\nAssociation Rules:")

print(association\_rules)

6. Write a program in python to implement Frequent-Pattern Tree algorithm.

import csv

import pyfpgrowth

transactions = []

with open('transaction\_table.csv', mode='r') as csv\_file:

reader = csv.reader(csv\_file)

for row in reader:

transactions.append(row)

# Finding the frequent patterns with min support threshold=0.5

FrequentPatterns = pyfpgrowth.find\_frequent\_patterns(transactions=transactions, support\_threshold=0.5)

print(FrequentPatterns)

# Generating rules with min confidence threshold=0.5

Rules = pyfpgrowth.generate\_association\_rules(patterns=FrequentPatterns, confidence\_threshold=0.5)

Rules

7.Write a program in python to print Pincer search algorithm.

from collections import defaultdict

import csv

from itertools import combinations

def read\_csv\_file(filename):

with open(filename, 'r') as file:

reader = csv.reader(file)

next(reader) # skip the first row (header)

transactions = []

for row in reader:

transactions.append([int(item) for item in row[1].split(',')])

return transactions

def pincer\_search(transactions, min\_sup):

# Initialize variables

itemsets = defaultdict(int)

frequent\_itemsets = []

n = len(transactions)

m = len(transactions[0])

start = 0

end = m - 1

# Phase 1: forward pincer movement

while start < n and end >= 0:

# Count the frequency of itemsets in the current range

count = defaultdict(int)

for i in range(start, n):

for j in range(end, -1, -1):

items = transactions[i][j:]

for k in range(len(items)):

itemset = tuple(sorted(items[:k] + items[k+1:]))

count[itemset] += 1

# Prune infrequent itemsets

infrequent\_itemsets = set(itemset for itemset, freq in count.items() if freq < min\_sup)

for itemset in infrequent\_itemsets:

del count[itemset]

# Add frequent itemsets to the result

frequent\_itemsets.extend(count.keys())

itemsets.update(count)

# Move the pincer

if end > 0:

end -= 1

else:

start += 1

# Phase 2: backward pincer movement

for i in range(n):

for j in range(m):

items = transactions[i][:j+1]

for k in range(len(items)):

itemset = tuple(sorted(items[:k] + items[k+1:]))

if itemset in itemsets and itemsets[itemset] >= min\_sup:

frequent\_itemsets.append(itemset)

# Generate association rules

rules = []

for itemset in frequent\_itemsets:

for i in range(1, len(itemset)):

for antecedent in combinations(itemset, i):

antecedent = tuple(sorted(antecedent))

consequent = tuple(sorted(set(itemset) - set(antecedent)))

support = itemsets[itemset] / float(n)

confidence = itemsets[itemset] / float(itemsets[antecedent])

lift = confidence / (itemsets[consequent] / float(n))

if confidence >= min\_conf:

rules.append((antecedent, consequent, support, confidence, lift))

return frequent\_itemsets, rules

# Example usage

filename = 'data.csv'

min\_sup = 3

min\_conf = 0.5

transactions = read\_csv\_file(filename)

frequent\_itemsets, rules = pincer\_search(transactions, min\_sup)

print("Frequent itemsets:")

for itemset in frequent\_itemsets:

print(list(itemset),end=',')

print("Association rules:")

for antecedent, consequent, support, confidence, lift in rules:

print(list(antecedent), "->", list(consequent), "Support:", round(support, 2), "Confidence:", round(confidence, 2), "Lift:", round(lift, 2))

8.Write a program in python to print Partition algorithm.

import pandas as pd

from mlxtend.frequent\_patterns import apriori, association\_rules

# Load the CSV file into a dataframe

df = pd.read\_csv('data.csv')

# Split the Items Purchased column into separate columns for each item

items = df['Items Purchased'].str.split(',', expand=True)

# Convert the columns to binary variables

df = pd.get\_dummies(items.stack()).sum(level=0)

# Define the number of partitions

num\_partitions = 2

# Split the dataframe into partitions

partitions = [df[i:i+len(df)//num\_partitions] for i in range(0, len(df), len(df)//num\_partitions)]

# Apply association rule mining to each partition

for i, partition in enumerate(partitions):

frequent\_itemsets = apriori(partition, min\_support=0.2, use\_colnames=True)

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.5)

print(f"Partition {i+1}:")

print(rules)

9.Write a program in python to implement K-means Algorithm.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

# data

data = np.array([

[7, 9],[21, 6],[14, 5], [21, 9],[7, 8],[9, 7], [14, 5],[15, 6],[16, 25]

])

# initialize k-means algorithm with 2 clusters

kmeans = KMeans(n\_clusters=2)

kmeans.fit(data)

# get the coordinates of the centroids and labels of the clusters

centroids = kmeans.cluster\_centers\_

labels = kmeans.labels\_

# plot the data points with different colors for each cluster and their respective labels

for i, point in enumerate(data):

plt.scatter(point[0], point[1], c=labels[i], cmap='rainbow', label=f'Cluster {labels[i]}')

plt.scatter(centroids[:,0], centroids[:,1], marker='o', color='black', s=100, label='Centroids')

plt.legend()

plt.show()