МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по практической работе №3
по дисциплине «Машинное обучение»

Студент гр. 1310 _____ Комаров Д. Е.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

Задание 1

Постановка задачи

Дан набор данных, представленный в таблице 1.

Таблица 1 – Набор данных для 1 задания

tid	itemset
t_1	$ABCD$
t_2	$ACDF$
t_3	$ACDEG$
t_4	$ABDF$
t_5	BCG
t_6	DFG
t_7	ABG
t_8	$CDFG$

1) Предположив, что минимальный уровень поддержки равен 3 / 8. Продемонстрируйте, как алгоритм Apriori перебирает данный набор данных.

2) Предположив, что минимальный уровень поддержки равен 2 / 8. Продемонстрируйте, как алгоритм FPGRowth перебирает данный набор данных.

Код программы

```
I={'A','B','C','D','E','F','G'}
D=[
    {'A','B','C','D'},
    {'A','C','D','F'},
    {'A','C','D','E','G'},
    {'A','B','D','F'},
    {'B','C','G'},
    {'D','F','G'},
    {'A','B','G'},
    {'C','D','F','G'}
    ]

def getSup(st,D):
    sup=0
    for d in D:
        if (st & d)==st:
            sup+=1
    return sup
def getNextC(L,I):
    C=set()
    for l in L:
       for i in I:
            if not(i in l):
```

```
                C=C|{(tuple(sorted(l|{i})))}
    C=[set(s) for s in C]
    return C

def apriori(D,I,minsup):
    F=[]
    C=[{i}for i in I]
    while C:
        L=[c for c in C if getSup(c,D)>=minsup]
        F.extend(L)
        C=getNextC(L,I)
    return F

def updateTree(R,d,mx):
    if not d: return
    for r in R:
        if r[0]==d[0]:
            r[1]+=mx
            updateTree(r[2],d[1:],mx)
            break
    else:
        R.append([d[0],mx,[]])
        updateTree(R[-1][2],d[1:],mx)

def inspectTreeDwn(R,elm):
    L=[]
    r2r=[]
    for r in R:
        if r[2]:
            for i in inspectTreeDwn(r[2],elm):
                i[0]=[r[0]]+i[0]
                L.append(i)
        else:
            if r[0]==elm:
                L.append([[],r[1]])
                r2r.append(r)
    for r in r2r:
        R.remove(r)
    return L


def FPGPrepareData(D,I,minsup):
    Idic={i:sum(1 for d in D if i in d) for i in I}
    Idic={i:Idic[i] for i in Idic if Idic[i]>=minsup}
    Idic=sorted(Idic,reverse=True,key=lambda a:Idic[a])
       Dord=[sorted([elm for elm in d if elm in Idic],key=lambda
a:Idic.index(a))for d in D]
    return Dord,Idic

def makeCPB(FP,I):
    CPB=[]
    while I:
        curr,I=I[-1],I[:-1]
        L=inspectTreeDwn(FP,curr)
        L=[l for l in L if l[0]]
        CPB.append([curr,L])
    return CPB

def makeCFP(CBP):
    CFT=[]
    for cpb in CBP:
        cft=[]
        for i in cpb[1]:
```

```
            updateTree(cft,i[0],i[1])
        CFT.append([cpb[0],cft])
    return CFT

def countTree(tree,minsup):
    if not tree: return {},set()
    dic,ptrns={},set()
    for t in tree:
        dwnptrns=set()
        dic[t[0]]= (dic[t[0]]+t[1]) if (t[0] in dic) else t[1]
        if t[2]:
            dwndic,dwnptrns=countTree(t[2],minsup)
            for elm in dwndic:
                      dic[elm]=(dic[elm]+dwndic[elm])  if  (elm  in  dic)  else
dwndic[elm]
            ptrns|=dwnptrns|{tuple(elm) for elm in dic if dic[elm]>=minsup}
        if t[1]>=minsup:
            ptrns|={tuple(t[0])}
            if dwnptrns:
                for elm in dwnptrns:
                    ptrns|={tuple(sorted(set(elm)|{t[0]}))}
    ptrns|={tuple(elm) for elm in dic if dic[elm]>=minsup}
    return dic,ptrns


def FPGrowth(D,I,minsup):
    D,I=FPGPrepareData(D,I,minsup)
    FP=[]
    for d in D:
        updateTree(FP,d,1)
    CPB=makeCPB(FP,I)
    CFT=makeCFP(CPB)
    F=set()
    for cft in CFT:
        dic,ptrns=countTree(cft[1],minsup)
        ptrns=[tuple(sorted(set(p)|{cft[0]}))for p in ptrns]
        ptrns.append(tuple(cft[0]))
        ptrns={tuple(sorted(p))for p in ptrns}
        F|=ptrns
    return [set(f)for f in F]

print(apriori(D,I,3))
print(FPGrowth(D,I,2))
```

Результат выполнения

Выполним алгоритм Apriori по шагам для минимальной поддержки 3. На первом шаге алгоритм выбирает все элементы, частота которых больше или равна минимальной поддержке. Такие элементы: [{'G'}, {'C'}, {'D'}, {'A'}, {'F'}, {'B'}]. На следующем шаге формируются пары из данных элементов и из них отбираются те, частота которых больше или равна минимальной поддержке. Такие пары: [{'F', 'D'}, {'C', 'G'}, {'A', 'C'}, {'C', 'D'}, {'A', 'D'}, {'G', 'D'}, {'A', 'B'}]. Следующим шагом к каждой паре добавляется по одному элементу, считается частота для троек и отбираются тройки, частота которых

больше или равна минимальной поддержке. Такая тройка всего одна: [{'A', 'C', 'D'}]. Следующим шагом формируются четверки из этой тройки и еще одного элемента и считается их частота. Поскольку не находится ни одной четверки, частота которой была бы больше или равна минимальной поддержке, работа алгоритма прекращается. По итогу работы алгоритма был получен список из следующих часто встречающихся наборов объектов: [{'G'}, {'A'}, {'C'}, {'B'}, {'F'}, {'D'}, {'A', 'D'}, {'C', 'D'}, {'C', 'A'}, {'F', 'D'}, {'B', 'A'}, {'D', 'G'}, {'C', 'G'}, {'C', 'A', 'D'}].

Выполним алгоритм FPGRowth по шагам для минимальной поддержки 2. Выполнение алгоритма начинается с того, что все объекты сортируются в порядке, обратном их частоте, отрекаются объекты, частота которых меньше минимальной поддержки и данные в исходном наборе сортируются в том же порядке, что и объекты. Объекты были отсортированы в следующем порядке: ['D', 'A', 'G', 'C', 'F', 'B']**.** Отсортированные данные:

1) ['D', 'A', 'C', 'B'],
2) ['D', 'A', 'C', 'F'],
3) ['D', 'A', 'G', 'C'],
4) ['D', 'A', 'F', 'B'],
5) ['G', 'C', 'B'],
6) ['D', 'G', 'F'],
7) ['A', 'G', 'B'],
8) ['D', 'G', 'C', 'F'].

После сортировки строится FP-дерево. Данные вносятся в дерево в отсортированном порядке. Если элемент имеет следующий за ним в качестве листа, ног у листа увеличивается счетчик на 1, иначе создается новый лист. Полученное дерево представлено на рисунке 1.

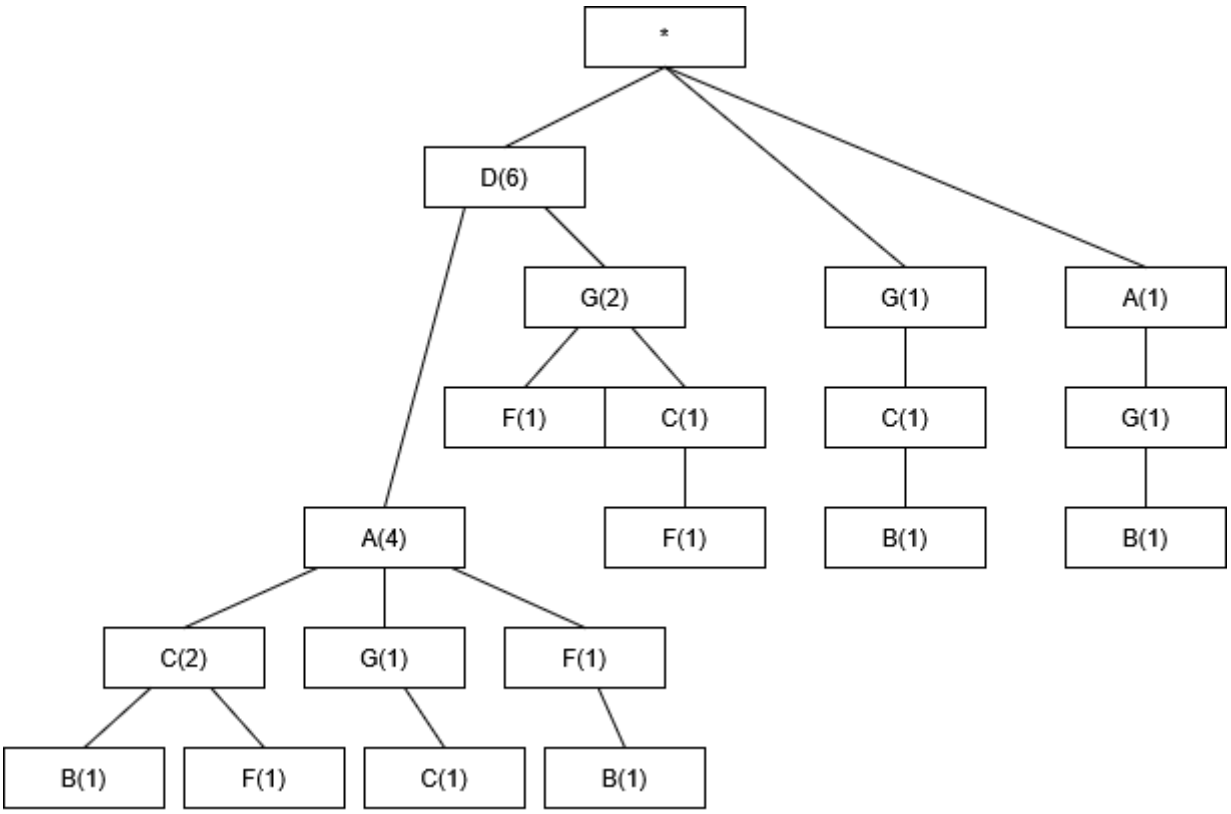

Рисунок 1 – FP-дерево

После того как дерево построено, для каждого из часто встречающихся объектов ищутся пути до них в FP-дереве. После того как путь был найден, он удаляется из FP-дерева. Были получены следующие наборы путей:

'B': [['D', 'A', 'C'], 1], [['D', 'A', 'F'], 1], [['G', 'C'], 1], [['A', 'G'], 1];

'F': [['D', 'A', 'C'], 1], [['D', 'A'], 1], [['D', 'G'], 1], [['D', 'G', 'C'], 1];

'C': ['D', 'A'], 2], [['D', 'A', 'G'], 1], [['D', 'G'], 1], [['G'], 1];

'G': [['D', 'A'], 1], [['D'], 2], [['A'], 1];

'A': [['D'], 4];

'D': [].

При помощи полученных путей строятся условные FP-деревья. Условные FP-деревья для каждого объекта представлены на рисунке 2.

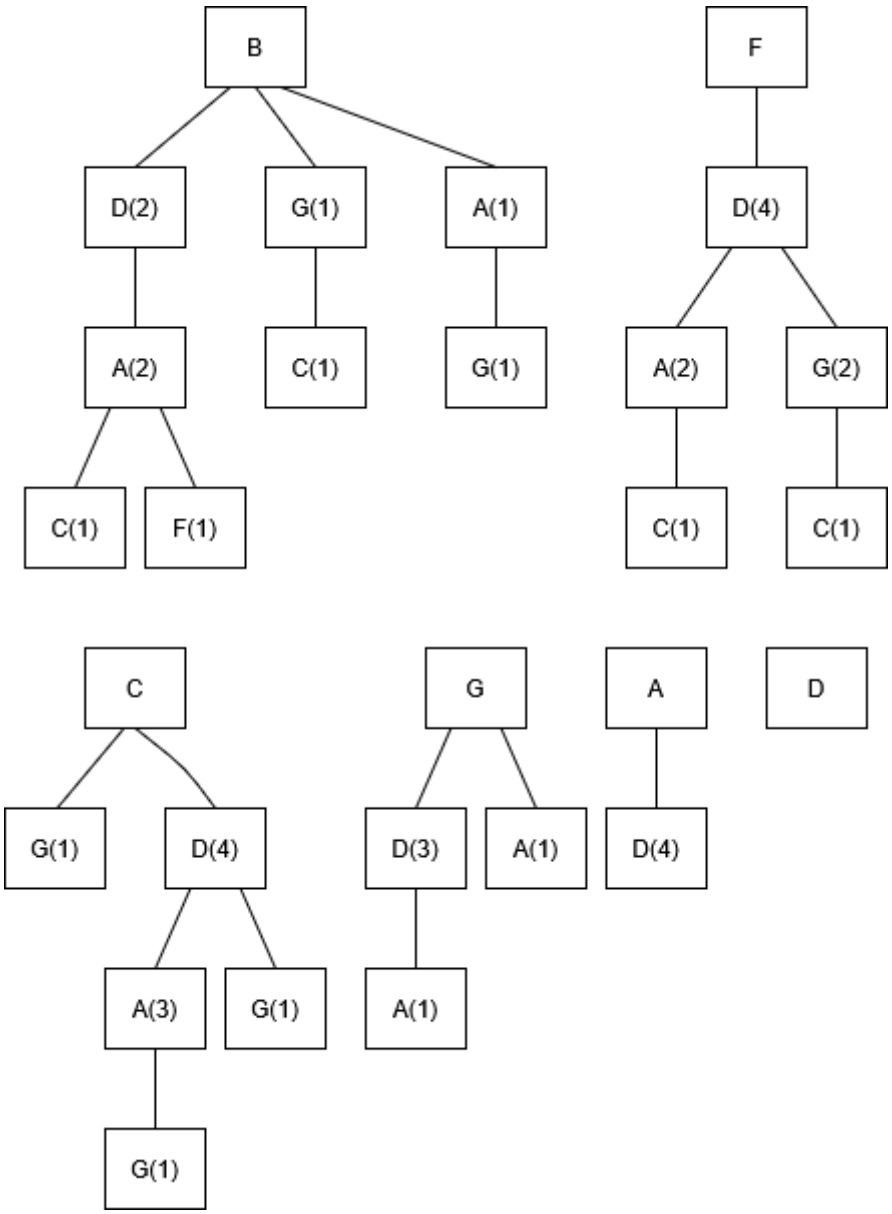

Рисунок 2 – Условные FP-деревья

Далее при помощи обхода условных FP-деревьев ищутся часто встречающиеся наборы. Для минимальной поддержки 2 были получены наборы: [{'A', 'D', 'F'}, {'G', 'A'}, {'A', 'B'}, {'G', 'C'}, {'G', 'F'}, {'A'}, {'B', 'D'}, {'D'}, {'B', 'C'}, {'C'}, {'D', 'F'}, {'G', 'D', 'C'}, {'A', 'F'}, {'G', 'D', 'F'}, {'B'}, {'F', 'C'}, {'G', 'B'}, {'G'}, {'A', 'D'}, {'A', 'C'}, {'D', 'C'}, {'F', 'D', 'C'}, {'A', 'D', 'C'}, {'G', 'D'}, {'F'}, {'A', 'B', 'D'}].

Задание 2

Постановка задачи

На рисунке 3 представлена классификация различных продуктов. Каждый лист дерева — это конкретный продукт, внутренний узел дерева представляет категорию продукта более верхнего уровня.

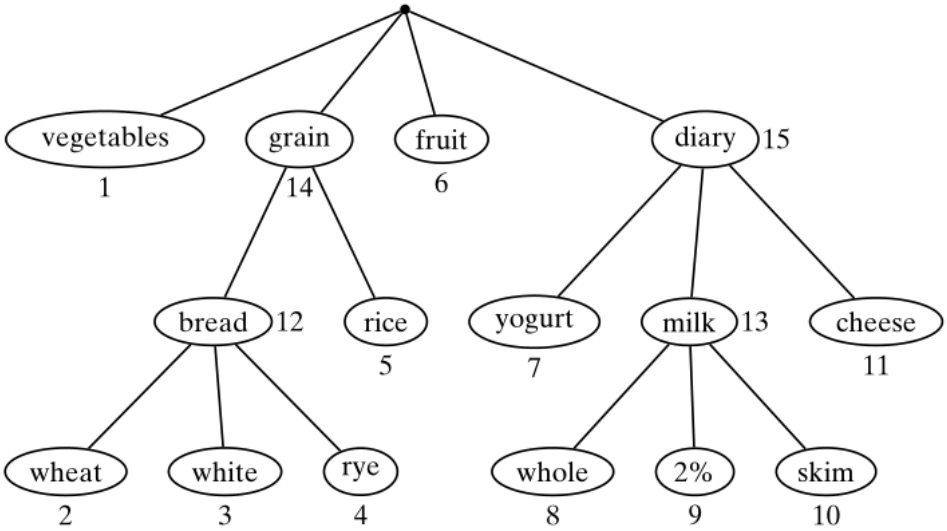

Рисунок 3 – Классификация различных продуктов

Был получен набор данных, представленный в таблице 2.

Таблица 2 – Набор данных для 2 задания

tid	itemset
1	2 3 6 7
2	1 3 4 8 11
3	3 9 11
4	1 5 6 7
5	1 3 8 10 11
6	3 5 7 9 11
7	4 6 8 10 11
8	1 3 5 8 11

Необходимо выполнить следующие задания:

1) Вычислить, каков размер области поиска наборов элементов, если ограничиваться только наборами, состоящими из простых элементов?

2) Предположив, что минимальный уровень поддержки = 7/8, необходимо найти все часто встречающиеся наборы элементов, состоящие

только из элементов высокого уровня в таксономии. Если в транзакции появляется простой элемент, предполагается, что все его предки высокого уровня также присутствуют в транзакции.

Код программы

```
I={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
D=[
    {2,3,7,6},
    {1,3,4,8,11},
    {3,9,11},
    {1,5,6,7},
    {1,3,8,10,11},
    {3,5,7,9,11},
    {4,6,8,10,11},
    {1,3,5,8,11}
    ]

def getSup(st,D):
    sup=0
    for d in D:
        if (st & d)==st:
            sup+=1
    return sup
def getNextC(L,I):
    C=set()
    for l in L:
       for i in I:
            if not(i in l):
                C=C|{(tuple(sorted(l|{i})))}
    C=[set(s) for s in C]
    return C

def apriori(D,I,minsup):
    F=[]
    C=[{i}for i in I]
    while C:
        L=[c for c in C if getSup(c,D)>=minsup]
        F.extend(L)
        C=getNextC(L,I)
    return F

for d in D:
    if ({2,3,4}&d)!=set():
        d|={12}
    if ({8,9,10}&d)!=set():
        d|={13}
    if ({12,5}&d)!=set():
        d|={14}
    if ({7,13,14}&d)!=set():
        d|={15}
print(apriori(D,I,7))
```

Результат выполнения

1) Если ограничится наборами, состоящими из простых элементов, которых 11, то размер области поиска наборов элементов может быть вычислен как

$$2^{11} = 2048.$$

2) Для минимальной поддержки 7 были получены следующие наборы: [{12}, {14}, {15}, {12, 15}, {14, 15}, {12, 14}, {12, 14, 15}].