
بسم الله الرحمن الرحیم

دانشگاه :شهید باهنر کرمان

دانشکده: فنی مهندسی

بخش : مهندسی کامپیوتر

موضوع : پیاده سازی الگوریتم IPHM مقاله

درس : داده کاوی

استاد درس: جناب آقای دکتر قاضی زاده

دانشجو:سوگند سماوی 99405044

پاییز 1403

```python
#Algorithm3
#شبه کد متناظر با هر الگوریتم در مقاله آورده شده است#
def Algorithm3(P , iUL_p , iULs , minutil , maxper , EUCS , sorted_list):
    PHUIs = []
    for iUL in iULs:

        if sum(i[1] for i in iUL[1]) >= minutil and iUL[2] <= maxper:

            PHUIs.append(iUL[0])

        if sum(i[1] for i in iUL[1]) + sum(i[2] for i in iUL[1]) >= minutil and iUL[2] <= maxper:
            EOf_Px_iULs = []
            for iUL2 in iULs:
                if sorted_list.index(iUL2[0][-1]) > sorted_list.index(iUL[0][-1]):  #  x < y (y ≻ x) # i*
                    target_pair = tuple([iUL[0][-1], iUL2[0][-1]])

                    c=next((value for pair, value in EUCS if pair == target_pair), None)
                    Px_y=[]
                    if c >= minutil:
                        Px_y.append(iUL[0][-1])
                        Px_y.append(iUL2[0][-1])

                        iUL_Px_y = Algorithm4(iUL,iUL2,  iUL_p ,6  )
                        EOf_Px_iULs.append(iUL_Px_y)

            P = P + [item for item in iUL[0] if item not in P]
            ph=Algorithm3(P, iUL, EOf_Px_iULs, minutil, maxper, EUCS ,sorted_list )
            if len(ph)>0:
                PHUIs.append(ph)
    return PHUIs
```

ابتدا الگوریتم 3 که در مقاله نیز شبه کد آن وجود دارد را پیاده سازی میکنیم. این الگوریتم ابتدا بررسی میکند که یک مجموعه قلم میتواند مجموعه قلم متنائبی و با سود بالا باشد یا خیر. اگر بود در خروجی نمایش داده میشود. سپس بررسی میشود که یک مجموعه قلم را میتوان گسترش داد یا خیر و اگر امکان پذیر بود هر بار یک قلم به مجموعه اضافه میشود و سپس بررسی میشود PHUI هست یا خیر و این کار تا زمانی که دیگر نتوان یک مجموعه قلم تناوبی سودمند ساخت ادامه پیدا میکند.

```python
#Algorithm4

def Algorithm4(iulPx, iulPy, iulP, LD):
    iulPxy = [[], [], 0]
    lastTid = 0
    for item in iulPx[0] + iulPy[0]:
        if item not in iulPxy[0]:
            iulPxy[0].append(item)

    for ex in iulPx[1]:
        for ey in iulPy[1]:
            if ex[0] == ey[0]:
                if len(iulP)>0:
                    if len(iulP[1]) > 0:
                        for e in iulP[1]:
                            if ex[0] == e[0]:
                                exy = [ex[0], ex[1] + ey[1] - e[1], ey[2]]
                else:
                    exy = [ex[0], ex[1] + ey[1], ey[2]]

                iulPxy[2] = max(ex[0] - lastTid, iulPxy[2])
                lastTid = ex[0]
                iulPxy[1].append(exy)


    iulPxy[2] = max(LD + 1 - lastTid, iulPxy[2])

    return iulPxy
```

الگوریتم 4 در الگوریتم 3 استفاده شده است که به وسیله آن میتوان iul حاصل از یک مجموعه قلم گسترش یافته را محاسبه کرد ، در الگوریتم 3 نیز برای اینکه بتوان یک مجموعه را به عنوان phui به حساب آورد یا بررسی کند که این مجموعه قلم قابلیت گسترش دارد یا خیر از مقادیر iutil و rutil موجود در iul ها استفاده میشود.

```python
#Algorithm2:

import numpy as np


def Algorithm2( Tnum , LUL):
    RuA=np.zeros(Tnum)
    for item in reversed(LUL):
        iul=item[1]
        for t in iul:
            t[2]=RuA[t[0]]
            RuA[t[0]]=RuA[t[0]] +t[1]
    return LUL
```

الگوریتم 2 برای به روزرسانی lul که مجموعه ای از iul ها است ، استفاده میشود و مقادیر iutil و rutil را به روزرسانی میکند.

از این الگوریتم زمانی که تراکنش جدید به تراکنش ها اضافه میشود استفاده میکنیم.

```
# Algorithm1
from collections import defaultdict
import numpy as np
D = [
    [1, [("a", 1), ("c", 1), ("e", 1)]],
    [2, [("b", 1), ("d", 2), ("f", 1), ("g", 1)]],
    [3, [("a", 1), ("d", 1), ("e", 2), ("f", 2), ("g", 1)]],
    [4, [("c", 3), ("e", 1), ("f", 1)]],
    [5, [("a", 1), ("b", 1), ("d", 1), ("e", 1), ("g", 2)]]
]
N= [
        [6, [("b", 2), ("c", 1), ("e", 3), ("g", 1)]],
        [7, [("a", 3), ("d", 2), ("f", 2)]
        ]]

unit_profit = {
    "a": 5,
    "b": 3,
    "c": 1,
    "d": 4,
    "e": 2,
    "f": 3,
    "g": 2,
}
I=[ item for item , key in unit_profit.items()]
print(I)
```

الگوریتم یک همان کد اصلی است که شامل الگوریتم های 2 و 3 و 4 هم میشود. در این قسمت ابتدا database ها را در کد تعریف کردیم که N شامل تراکنش های جدید و unit_profit شامل وزن یا همان قیمت هر قلم است.

```
def calculate_tu(transaction):
    return sum(unit_profit[item] * quantity for item, quantity in transaction)

twu = {}
def calculate_twu(D , twu):
    for tid, transaction in D:
        tu = calculate_tu(transaction)
        for item, _ in transaction:
            twu[item] = twu.get(item, 0) + tu

    return twu

twu = calculate_twu(D , twu)

item_positions = defaultdict(list)
last_tid_item ={}

for transaction_id, items in D:
    for item, _ in items:
        item_positions[item].append(transaction_id)

max_durations = {}

for item, positions in item_positions.items():
    last_tid_item[item]=positions[-1]
    if len(positions) > 1:
        # برای هر قلم، طول دوره‌ها را محاسبه می‌کنیم
        durations = [positions[i+1] - positions[i] for i in range(len(positions) - 1)]
        max_durations[item] = max(durations)
```

در این قسمت دو تابع برای محاسبه tu و twu نوشته شده است ، سپس طول دوره هر آیتم محاسبه شده و مقادیر ماکزیمم در max_duration ذخیره میشوند.

```
sorted_items = sorted(twu.items(), key=lambda x: (x[1], x[0]))
sorted_items = [item for item, _ in sorted_items]

deleted_items = [i for i in sorted_items if  (twu[i] < 23 or max_durations[i] > 2)]

sorted_items = [i for i in sorted_items if not (twu[i] < 23 or max_durations[i] > 2)]
```

بعد از محاسبه twu اقلام بر اساس آن و به صورت صعودی مرتب میشوند.

```
#.....................................................iul
def construct_iul(item, transactions, sorted_items, unit_profit):
    iul = []
    if item in sorted_items:
        for transaction in transactions:

            for idx, (itm, itm_count) in enumerate(transaction[1]):
                if itm == item:

                    item_profit = unit_profit[itm] * itm_count

                    after_item_profit = 0
                    if itm in sorted_items:
                        idx_in_sorted = sorted_items.index(itm)
                        after_items = sorted_items[idx_in_sorted + 1:]

                        for after_item in after_items:

                            for itm2, itm2_count in transaction[1]:
                                if itm2 == after_item:

                                    after_item_profit += unit_profit[itm2] * itm2_count
                    iul.append([transaction[0], item_profit, after_item_profit])
    return iul

     print(f"IUL for item {item}: {iul}")
```

در این قسمت تابع ساخت iul تعریف شده که هربار یک سه تایی جدید که شامل tid , iutil , rutil هست را برای مجموعه قلم به iul اضافه میکند.

```
def generate_lul(D, sorted_items, unit_profit , max_durations ):
    lul = []
    for item in sorted_items:
        x=[]
        x.append(item)
        period=max_durations[item]

        iul = construct_iul(item, D, sorted_items, unit_profit)
        if iul:
            lul.append([x, iul ,  period])
    return lul
```

در این تابع مجموعه iul ها در یک مجموعه تحت عنوان lul قرار میگیرند.

```
twu1={}
def calculate_eucs(D, sorted_items, unit_profit , twu1 , deleted_items ):
    twu = calculate_twu(D , twu1)

    eucs_list = []


    for i in range(len(sorted_items)):
        for j in range(i + 1, len(sorted_items)):  # ترکیب‌های دو به دو
            item1 = sorted_items[i]
            item2 = sorted_items[j]

            twu_combination = 0
            for tid, transaction in D:

                items_in_transaction = [itm for itm, _ in transaction]
                if item1 in items_in_transaction and item2 in items_in_transaction:

                    tu = calculate_tu(transaction)
                    for y in transaction:
                        if y[0] in deleted_items:
                            tu=tu-y[1]*(unit_profit[y[0]])

                    twu_combination += tu

            eucs_list.append(((item1, item2), twu_combination))

    return eucs_list
```

در این تابع یک ماتریس تحت عنوان EUCS ساخته میشود که twu حاصل برای هر دو قلم را نشان میدهد.

```
twu2={}
# print(calculate_eucs(D, sorted_items, unit_profit , twu2 , deleted_items))
P=[]
iUL_p=[]
lul=(generate_lul(D, sorted_items, unit_profit , max_durations))
# print(lul)
minutil=23
maxper=2
EUCS=calculate_eucs(D, sorted_items, unit_profit , twu1 , deleted_items)
PHUISS=Algorithm3(P , iUL_p , lul , minutil , maxper , EUCS , sorted_items)
print(PHUISS)
```

در این قسمت از الگوریتم سه استفاده شده تا PHUI های موجود در دیتابیس D استخراج شوند.

```
#-------------------------------------------------------------------
# for N dataBase
twu_new = calculate_twu(N , twu)
twu_new= {key: twu_new[key] for key in sorted_items if key in twu_new}

item_positions_N = defaultdict(list)

for transaction_id_N, items_N in N:
    for item_N, _ in items_N:
        if item_N in sorted_items:
            item_positions_N[item_N].append(transaction_id_N)

for i_N in sorted_items:
    item_positions_N[i_N].insert( 0 , last_tid_item[i_N])


max_durations_N = {}

for item_N, positions_N in item_positions_N.items():

    if len(positions_N) >= 1:

        durations_N = [positions_N[i+1] - positions_N[i] for i in range(len(positions_N) - 1)]

        max_durations_N[item_N] = max(durations_N)
for key in max_durations_N.keys():
    if key in max_durations:
        max_durations_N[key] = max(max_durations_N[key], max_durations[key])

sorted_items_new = sorted(twu_new.items(), key=lambda x: (x[1], x[0]))
sorted_items_new = [item for item, _ in sorted_items_new]
```

در این قسمت تراکنش های جدید به دیتابیس اضافه شدند پس طول دوره هر قلم به روز رسانی میشود.

```
deleted_items_new = [i for i in sorted_items_new if  (twu_new[i] < 23 or max_durations_N[i] > 2)]
sorted_items_new = [i for i in sorted_items_new if not (twu_new[i] < 23 or max_durations_N[i] > 2)]


lul_N = sorted(
    [x for x in lul if x[0][0] in sorted_items_new],
    key=lambda x: sorted_items_new.index(x[0][0])
)


for transaction in N:
    transaction_id = transaction[0]
    items = transaction[1]

    for item, count in items:

        for entry in lul_N:
            if entry[0][0] == item:

                profit = count * unit_profit[item]

                entry[1].append([transaction_id, profit, 0])
lul_N=Algorithm2( 8 , lul_N)
```

در این قسمت هم از الگوریتم 2 برای به روز رسانی lul استفاده شده است.

```
for Item in lul_N:
    Item[2]=max_durations_N[Item[0][0]]


twu2={}
DN=[]
DN.extend(D)
DN.extend(N)
deleted_items_new.extend(deleted_items)

EUCS_new=calculate_eucs(DN, sorted_items_new, unit_profit , twu2 , deleted_items_new)

print("_____")
PHUISS2=Algorithm3(P , iUL_p , lul_N , minutil , maxper , EUCS_new , sorted_items_new)

print(f"PHUIS for D+N database are :{PHUISS2}")
```

```
items are['a', 'b', 'c', 'd', 'e', 'f', 'g']
PHUIS for D database are :[[['a', 'e']], [['d', 'g']]]
_____
PHUIS for D+N database are :[[['e', 'a']], [['g', 'd']], ['a'], ['d']]
```

در آخر دوباره از الگوریتم 3 برای استخراج PHUI ها برای دیتابیس جدید که شامل تراکنش های قدیم و جدید است استفاده کردیم و مشاهده میشود که برای دیتابیس D مجموعه های [a , e] , [d , g] و برای دیتابیس D+N مجموعه های [a] , [d] , [a , e] , [d , g] به عنوانPHUI شناسایی شده اند.