Gyakori elemhalmazok

Peregi Tamás

BME-SZIT

2011. 03. 08.

Tartalom

Hol tartunk?

Egy érdekes történet

Az egyik áruházlánc komputereinek adataiból kiderült, hogy a késő este pelenkát vásárló vevők közül sokan sört is tettek a bevásárlókosarukba. Feltételezve, hogy az esti vásárlók valószínűleg férfiak, akiket a feleségük küld el pelenkáért, az áruházak a pelenkás és sörös polcokat közelebb tették egymáshoz. Ily módon ösztönözték az apukákat, hogy ne csak a mindennapi pelust, hanem saját söradagjukat is szerezzék be.

Hogyan tudunk ilyen dolgokat észrevenni? Erről szól a mai előadás.

Fogalmak

- $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ az elemek (vásárolható termékek) halmaza
- $\mathcal{T} = \langle t_1, t_2, \ldots, t_n \rangle$ a tranzakciók (vásárlások) sorozata, $t_i \subseteq \mathcal{I}$
- $I \subseteq \mathcal{I}$ támogatottsága azon tranzakciók száma, amelyeknek I részhalmaza. Jele: $supp(I)$
- I gyakori, ha $supp(I) \geq min\_supp$ (ahol min_supp adott korlát)
- I ritka, ha nem gyakori

Kérdés

Adott $\mathcal{I}$, $\mathcal{T}$ és min_supp esetén melyek $\mathcal{I}$ gyakori részhalmazai?

Méretek a gyakorlatban

- A gyakorlatban $|\mathcal{I}| \approx 10^6$ és $|\mathcal{T}| \approx 10^9$ is lehet.
- A kimenet mérete lehetséges, hogy $|\mathcal{I}|$-ben exponenciális!
- Szerencsére a gyakorlati alkalmazásokban a tranzakciók maguk viszonylag kicsik, és a gyakori elemhalmazok mérete (és száma) sem lesz kezelhetetlenül nagy.

Egy egyszerű algoritmus

- Sorban olvassuk be $\mathcal{T}$ elemeit, és minden tranzakció minden részhalmazára vizsgáljuk meg, hogy el van-e már tárolva a memóriában.
  - Ha nem, tároljuk el, és rendeljünk hozzá egy 1 kezdeti értékű számlálót.
  - Ha igen, növeljük meg a hozzárendelt számláló értékét.
- $\mathcal{T}$ végigolvasása után már az összes potenciális (legalább egyszer megjelenő) gyakori elemhalmaz támogatottsága ismert, így könnyedén kiszűrhetők a gyakoriak.

Probléma

Nincs elég memória - egy 60-elemű tranzakció részhalmazainak száma 10^{18} nagyságrendű, azaz csak a számlálók tárolására 10^6 TiB nagyságrendű memóriára lenne szükség.

Egy egyszerű algoritmus

Javítási ötletek

- Csak egy adott számnál kisebb elemszámú részhalmazokat tároljunk.
- A jelölteket és számlálóikat tároljuk szófában.

Problémák

- Nem kapunk meg minden gyakori elemhalmazt
- Túl sok hamis jelölt $\Longrightarrow$ még mindig túl nagy memóriaigény

Hol tartunk?

Alapötlet

Megfigyelés

Gyakori elemhalmaz minden részhalmaza gyakori.

Ötlet

Az $\ell + 1$ elemű jelölteket az ℓ elemű gyakori elemhalmazok páronkénti unióiból állítsuk elő.

Pszeudokód

ℓ := 0
J_ℓ := {$\emptyset$}
while J_ℓ nem üres **do**
  támogatottság_meghatározás($\mathcal{T}$, J_ℓ)
  GY_ℓ := gyakori_kiválogatás(J_ℓ, min_supp)
  $J_{\ell+1}$:= jelölt_előállítás(GY_ℓ)
  ℓ := ℓ+1
end while

Támogatottság meghatározása - triviális algoritmus

Minden jelölt támogatottságát külön határozzuk meg, megvizsgálva minden tranzakciót, hogy tartalmazza-e a jelöltet.

Probléma

Sok jelölt esetén lassú: végig kell nézni mindig a teljes tranzakciót.

Támogatottság meghatározása szófák segítségével

- A jelölteket szófában tároljuk.
- Minden jelölthöz (levélhez) egy számláló, amiben a támogatottságuk van.
- Egy tranzakció feldolgozásakor bejárjuk a fát
  - Ha egy d szintű csúcshoz a tranzakció j. elemén keresztül jutunk, akkor azokon az éleken megyünk tovább, amelyek címkéje $t[j']$, ahol $j < j' \leq |t| - \ell + d$.
  - ℓszintű levél elérésekor megnöveljük a számláló értékét.
- Az összes tranzakció feldolgozása után a gyakoriságok kiolvashatók a levelek számlálóiból.

Gyakori elemhalmazok kiválogatása

- Bejárjuk a szófát, és amelyik levél számlálója kisebb *min_supp*-nál, azt töröljük a fából.
- A gyakorinak talált elemhalmazokat kiírjuk a kimenetre.

Jelöltek előállítása

- Keressünk olyan ℓ hosszú gyakori I_1, I_2 elemhalmaz párokat, amikre
  - I_1 megelőzi lexikografikusan I_2-t
  - I_1 utolsó elemét elhagyva ugyanazt kapjuk, mint I_2 utolsó elemét elhagyva
- A pár unióját képezve vizsgáljuk meg, hogy minden részhalmaza gyakori-e. Ehhez elég az ℓ elemű részhalmazokat megvizsgálni.
- Ez szófákkal könnyen megoldható: az előző fában a testvérlevelek segítségével kell új jelölteket generálni.

Rendezés hatása az algoritmusra

- Eddig mindig feltételeztük, hogy az elemek a jelöltekben és a tranzakciókban valahogy sorrendezve vannak.
- A szófa alakja (és pontjainak száma) függ a rendezéstől. Hogy érdemes megválasztani a sorrendet, hogy minimális memóriát használjunk?

Tétel

A minimális méretű szófa meghatározásának problémája NP-nehéz.

Heurisztikák a rendezésre 1.

Ötlet

Rendezzük az elemeket gyakoriság szerint csökkenő sorrendbe!

- Jónak tűnik, mert a gyakori elemek kerülnek a sorozatok elejére, és így a gyakori közös prefixeket csak egyszer tárolja a fa.
- Tapasztalatok alapján tipikusan kisebb fa, mint a növekvő rendezés, vagy véletlenszerű rendezések esetén.

Heurisztikák a rendezésre 2.

Ötlet

Rendezzük az elemeket mégis inkább gyakoriság szerint növekvő sorrendbe!

- Nagyobb memóriaigény (tipikusan)
- Kevesebb él indul ki egy pontból
- A tranzakcióban a gyakori elemek hátrakerülnek, így a támogatottság-meghatározáskor nagyrészt a tranzakció végén kapunk csak találatokat, így kevesebb irányba mehetünk utána csak (mivel figyelnünk kell rá, hogy maradjon elég elem még az ℓ-elemű halmazhoz) $\Longrightarrow$ a fa kisebb részét járjuk csak be, gyorsabb a futás

Zsákutca nyesés

Ötlet

Ha egy csúcsból elérhető összes levelet töröltük, a csúcs felesleges, így ő is törölhető.

- A jelöltek előállítása során egyszerűen töröljük azokat a csúcsokat, akikből nem tudunk új jelöltet generálni.
- A gyakoriak kiválogatása során minden csúcs törlésekor töröljük a szülőjét, ha ő volt az egyetlen gyereke.

Tranzakciók szűrése 1.

Ötlet

Minden tranzakcióból törölhetők a ritka elemek.

Ötlet

Az ℓ. iterációban t törölhető, ha $|t| \leq \ell$. (Nem tartalmaz későbbi jelöltet.)

Ötlet

Ha egy tranzakció nem tartalmaz jelöltet, törölhető.

Tranzakciók szűrése 2.

Előző ötlet finomítása

Egy tranzakcióból törölhetők azok az elemei, amelyek nem elemei egyetlen általa tartalmazott jelöltnek sem.

Előző ötlet további finomítása

Egy tranzakcióból törölhetők azok az elemei, amelyek nem elemei legalább ℓ általa tartalmazott jelöltnek sem.

Equisupport nyesés

Megfigyelés

Legyen $X \subset Y \subseteq \mathcal{I}$. Ha $supp(X) = supp(Y)$, akkor $supp(X \cup Z) = supp(Y \cup Z)$ minden $Z \subseteq \mathcal{I}$-re.

- Ha egy jelölt támogatottsága megegyezik a prefixének támogatottságával, akkor alatta ugyanaz a fa épülne ki, mint a szülő alatt $\Longrightarrow$ felesleges „rendesen megcsinálni”
- Vegyük fel a kiegészítő elemet a szülő equisupport halmazába
- Amikor egy gyakori elemhalmazt kiírunk, vele együtt kiírjuk az E halmaz részhalmazaival vett unióit is. (E a saját és prefixeinek equisupport halmazainak uniója)

Equisupport nyesés implementáció

- A ritka elemek törlésekor töröljük az olyan jelölteket, akiknek a támogatottsága megegyezik a szülőjével, és hozzáfűzzük a szülő equisupport halmazához.
- Felfoghatók hurokélként
  - Támogatottság-meghatározáskor nem kell figyelembe venni
  - Jelölt-előállításkor az ellenőrzésnél figyelembe kell venni
- Zsákutca-nyeséskor a mélységbe bele kell érteni a prefixek és saját equisupport halmazok méretét

Hol tartunk?

Alapötlet

- Ha az Apriori „szélességi bejárás” volt, akkor ez „mélységi” lesz.
- Mindig egy elemet állít elő, és azonnal meghatározza a támogatottságát.
- Egy elem TID-halmaza: azon tranzakciók sorszámai, amelyek tartalmazzák az adott elemhalmazt.
  - A TID-halmaz mérete a támogatottság
  - Jelölt TID-halmaza a generátorok TID-halmazainak metszete

Az algoritmus előkészítése

- Gyűjtsük ki a gyakori elemeket
- Építsük fel a gyakori elemek TID-halmazait
- Hívjuk meg az ECLAT-SEGÉD eljárást, paraméterül az egy elemű gyakori elemhalmazok halmazát átadva.

A jelöltgenerálás ugyanaz, mint az Apriori esetén.
GY_P a P prefixű, annál eggyel hosszabb gyakori elemhalmazokat tartalmazza

```
ECLAT_SEGÉD(GY_P):
  for all gy∈GY_P do
    for all gy'∈GY_P, gy<gy' do
      j:=gy ∪ gy'
      j.TID:= gy.TID ∩ gy'.TID
      if |j.TID| ≥ min_supp then
        GY_gy := GY_gy∪{j}
      end if
    end for
    if |GY_gy| ≥ 2 then
      GY:= GY ∪ GY_gy ∪ ECLAT_SEGÉD(GY_gy)
    else
      GY:= GY ∪ GY_gy
    end if
  end for
  return GY
```

Elemzés

Megfigyelés

Az Eclat legalább annyi jelöltet állít elő, mint az Apriori, ilyen szempontból tehát rosszabb annál.

Előnyök az Apriorihoz képest

- A jelöltek TID-halmazának előállítása egyszerű és gyors
- Ahogy egyre mélyebbre jutunk, a TID-halmazok mérete csökken $\Longrightarrow$ a támogatottság meghatározásának ideje is csökken

Hogyan rendezzünk?

- A cél az, hogy minél kevesebb olyan jelölt legyen, aki az Aprioriban nem kerül elő, vagyis van ritka részhalmaza.
- A jelöltek prefixe gyakori, így akkor nagy az esélye a jelöltnek, ha a leggyakoribb elemet nem tartalmazza a prefix $\Longrightarrow$ gyakoriság szerint növekvő sorrendbe érdemes rendezni.

Hol tartunk?

Alapfogalmak

Szűrés

A $\mathcal{T}$ bemenet szűrésén azt értjük, hogy a benne szereplő tranzakciókból elhagyjuk a ritka elemeket. Az így kapott halmazt jelöljük $\mathcal{T}^*$-gal.

Vetítés

A $\mathcal{T}$ bemenet P elemhalmazra vetítését $(\mathcal{T}|P)$ úgy kapjuk, hogy vesszük a P-t tartalmazó tranzakciókat, és elhagyjuk belőlük P-t.

Algoritmus

A rekurzív FP_GROWTH_SEGÉD eljárás egy P elemhalmazt és egy erre vetített $\mathcal{T}$ tranzakcióhalmazt vár paraméterül.

- Meghatározzuk a $\mathcal{T}$-ben szereplő elemek támogatottságát, és kiválasztjuk a gyakoriakat
- Megszűrjük $\mathcal{T}$-t
- Minden gy gyakori elem esetén
  - $P \cup gy$ gyakori elemhalmazt kiírjuk a kimenetbe
  - Meghívjuk az eljárást rekurzívan $P \cup gy$ és $\mathcal{T}^*|gy$ paraméterekkel
  - Kitöröljük gy-t $\mathcal{T}^*$-ból

Megvalósítás

A megvalósításhoz FP-fát használunk

- Bővített szófa
- A tranzakciókat tároljuk el benne
- Nem csak a levélelemekben van számláló (hány tranzakciónak prefixe az adott csúcshoz tartozó halmaz)
- Azonos címkéjű csúcsok láncolt listaként összekötve, a lánc első eleme egy fejléctáblában van eltárolva
- Gyakoriság szerint csökkenő sorrendben építjük fel a fát

FP-fa használata 1.

FP-fa felépítése

Ugyanúgy, mint egy egyszerű szófát, csak minden számlálót megnövelünk „útközben”. Új csúcs felvétele esetén belefűzzük a megfelelő lista elejére.

Támogatottság meghatározása

Végighaladunk az adott elemhez tartozó láncolt listán, és összadjuk a számlálók értékét

FP-fa használata 2.

Elem törlése

Ha a gyakori elemeken gyakoriság szerint növekvő sorrendben megyünk végig az algoritmus során, és végezzük a vetítés-rekurzió-törlés lépéseket, akkor mindig leveleket kell törölnünk, amit könnyedén megtehetünk.

Egy elemre vetített tranzakcióhalmaz meghatározása

Amennyiben olyan elemre vetítünk, amelyik csak levelekben fordul elő (ez teljesül a növekvő sorrend esetén), a keresztéleken végighaladva az aktuális csúcstól a gyökérig vezető út pontjai alkotják a vetített tranzakciókat. Ezekből egyszerűen egy új fát építünk.

FP-fa használata 3.

Rekurzió leállása

Ha az FP-fa már csak egyetlen útból áll, akkor álljunk le, és írjuk ki az összes részhalmazát. Egy részhalmaz támogatottsága a kegmélyebben lévő csúcsának számlálója.

Köszönöm a figyelmet!