Valós időben választ adó egészségügyi profil, mint többdimenziós megszorítás mátrix, alapján élelmiszert szűrő domain specifikus algoritmus

Kusper Gábor[1], Márien Szabolcs[2], Kovács Emőd[3], Kovács László[4]

Eszterházy Károly Főiskola[1,3], Wit-Sys Zrt.[2], Miskolci Egyetem[4]

gkusper@aries.ektf.hu[1], Szabolcs.Marien@wit-sys.hu[2], emod@aries.ektf.hu[3], kovacs@iit.uni-miskolc.hu[4]

Absztrakt

Az eFilter projekt célja egy olyan informatikai rendszer felállítása, amely egészségügyi adatok alapján szűri az élelmiszerek listáját, amelyet a felhasználók szeretnének elfogyasztani. Itt az élelmiszer lehet egy nagyon egyszerű, pl. liszt, vagy akár egy nagyon összetett, pl. sajtos makaróni, is. Az egészségügyi adatokat egy egészségügyi profilban tároljuk. Ez tartalmazza az ételérzékenységeket, az allergiákat, diétákat és egyéb étkezésnél figyelembe veendő adatokat. Ugyanakkor ezeket az adatokat számszerűsítve tároljuk. Nem azt tároljuk, hogy a felhasználónak mogyoró allergiája van, hanem hogy a megengedett napi mogyoró bevitel 0.0 és 0.0 közt van, tehát megszorításként. Így az egészségügyi profil egy többdimenziós megszorítás mátrix lesz. Ez alapján kell minél gyorsabban megvizsgálni az ételeket, hogy fogyaszthatók-e. Ehhez ismerni kell az étel összetételét. Ezen adatforrás megszerzésével és hitelességével egy másik cikkben foglalkozunk. Ebben a cikkben a megszorítás mátrixot felhasználó algoritmusokat írunk le. A legfejlettebb algoritmus egy indexelési eljárás segítségével valós időben képes megvizsgálni, hogy fogyasztható-e egy élelmiszer. Ehhez az élelmiszer adatbázist indexeljük az összetevői alapján. Az indexelés megértéséhez vegyük a mogyoró tartalmat. Az első szintű index azokat az élelmiszereket tartalmazza, ahol a mogyoró tartalom nulla. A második szintű index azokat, ahol a mogyoró tartalom, egy milligramm (mg), azaz kettő a nulladikon. Az n. index azokat, ahol a mogyoró tartalom 2^(n-1) mg. Így összetevőként 18 index 0,1 kg-os tartalom leírására is alkalmas, a mg-os kategóriában nagyon pontosan, a kg-os kategóriában hozzávetőlegesen. Így a valós számok összehasonlítása visszavezethető indexelésre. Ezt az algoritmust a fenti feladattól elvonatkoztatva is vizsgáljuk.

1. Bevezetés

Ebben a cikkben olyan egyszerű, majd egyre kifinomultabb algoritmusokat közlünk, amelyek egy ételről, amelynek ismerjük az összetevőit, egy az összetevőkre vonatkozó megszorítás halmaz alapján eldöntik, hogy megfelelnek-e a megszorításoknak vagy sem, ehetem-e az ételt vagy sem.

A cikk fő motivációja az eFilter projekt, amelyet részletesen bemutatunk a következő fejezetben. Az eFilter projekt az EgerFood projekt továbbgondolása. Ez EgerFood projektben létrehoztunk egy olyan információs rendszert [EF2007a - EF2007c, EF2008a - EF2008c], amelynek segítségével a cégek le tudják tárolni, akár egyedi termékenként, hogy milyen összetevőkből készült az élelmiszer, az összetevőket ki szállította, mikor és hogyan használták azokat fel.

Az eFilter projekt szemszögéből ebből az a fontos, hogy néhány élelmiszerről egészen pontosan tudjuk, mit tartalmaz. Más adatforrást is használtunk az itt közölt algoritmusok vizsgálatához. Fő forrásunk a http://www.ars.usda.gov/Services/docs.htm?docid=20959 linkről letölthető, az United States Department Of Agriculture szervezet által közölt élelmiszer adatbázis.

Cikkünk két jól különhatárolható részre tagozódik. Az első részben egy-egy étel vizsgálatára alkalmas algoritmusokat közlünk. A 3. fejezetben a nem dinamikus algoritmusokkal foglalkozunk. Ezekben a megszorításokhoz nincs idő társítva. A 4. fejezetben a dinamikus algoritmusok kerülnek sorra. Itt minden megszorításnál meg van adva, hogy az melyik étkezésre vagy napszakra vonatkozik.

A második nagy részben az IMEE algoritmussal foglalkozunk, amely a cikk fő eredményének tekinthető. Ez az adatbázis kezelésben jól ismert bitmap indexelést alkalmazza egy élelmiszer adatbázison. Az indexelés segítségével a megszorítások visszavezethetők index vizsgálatra. Mivel csak kezelhetően sok indexet szabad felállítani, a visszavezetés csak bizonyos pontossággal lehetséges. Az 5. fejezetben ismertetjük az IMEE algoritmust, alkalmazásának lehetőségeit. A 6. fejezetben az IMEE algoritmus vizsgálati teszt eredményeit ismertetjük. A 7. fejezetben összefoglaljuk munkánkat.

2. Az eFilter projekt bemutatása

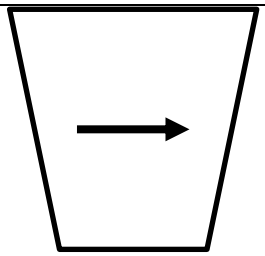
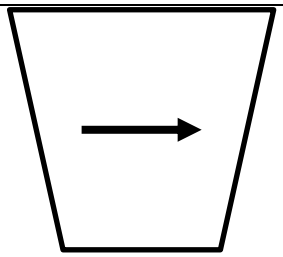
Az eFilter projekt célja egy olyan informatikai rendszer felállítása, amely egészségügyi adatok alapján szűri az élelmiszerek listáját, amelyet a felhasználók szeretnének elfogyasztani. Itt az élelmiszer lehet egy nagyon egyszerű, pl. liszt, vagy akár egy nagyon összetett, pl. sajtos makaróni, is. Az egészségügyi adatokat egy egészségügyi profilban tároljuk. Ez tartalmazza az ételérzékenységeket, az allergiákat, diétákat és egyéb étkezésnél figyelembe veendő adatokat. Ugyanakkor ezeket az adatokat számszerűsítve tároljuk. Nem azt tároljuk, hogy a felhasználónak mogyoró allergiája van, hanem hogy a megengedett napi mogyoró bevitel 0.0 és 0.0 közt van, tehát megszorításként. Így az egészségügyi profil egy többdimenziós megszorítás mátrix.

Az eFilter rendszert a KMOP-1.1.1-09/1-2009-0053 számú pályázat keretében a hozzuk létre.

2.1. Működési modell

Minden használati eset két alapvető működési módra vezethető vissza. A kétféle használati módot összefoglaló ábra:

Bő lista	Egészségügyi profil	Szűk lista

Étkeztető szervezet menü listája Bolt élelmiszer listája Étel rendelésnél étlap Hol kapható a fogyasztható termék?	 SZŰRÉS	Fogyasztható menük listája Boltban kapható fogyasztható élelmiszerek listája Fogyasztható ételek listája Fogyasztható terméket áruló boltok listája
Kérdés	Egészségügyi profil	Válasz
Megvásárolt élelmiszer fogyasztható-e? Vásárlásnál megerősítés, hogy az adott termék fogyasztható-e?	 ELLENŐRZÉS	Igen / Nem + Megvásárolt élelmiszerről bő információ Igen/Nem + Megvásárolt élelmiszerről bő információ

A szűrés során a bő lista minden egyes elemére az egészségügyi profilt alkalmazva eldönthető, hogy az adott termék megfelel-e a profilban megadott megszorításoknak vagy sem. Továbbá a fogyasztható termékekhez való hozzájutást segítő információk kérhetőek. Az ellenőrzés során egy termékről eldönthető, hogy megfelel-e az egészségügyi profilban megadott megszorításoknak vagy sem. Továbbá a termékkel kapcsolatos információk kérhetők le.

2.2. Adatmodell

Az egészségügyi profil tulajdonságai:

- Az egészségügyi profil lehet személyes, azaz fogyasztóhoz köthető vagy sablon, amely fogyókúra és egyéb betegségek esetén általános megkötések tételére használható.
- A személyes és a sablon egészségügyi profil is kor-, nem- és súlyfüggőek. Ezek leíró attribútumként az egészségügyi profil entitás részei.
- Az egészségügyi profilok strukturálhatók, azaz a már meglévő profilokból és azok szabályaiból építkezni lehet. A meghatározott egészségügyi profil sablonok felhasználhatók személyes profilok esetén. Például egy előre meghatározott betegséghez kötött egészségügyi profil beköthető egy személyes profilhoz, amelyet az adott beteg egyedi szabályaival tovább lehet finomítani.

Egészségügyi profil szabály:

- Megszorítások, amelyek irányai a következők lehetnek:
  - tiltás,
  - nem javasolt,
  - javasolt,
  - erősen javasolt,

- amelyek megadása mennyiségi korláttal történik. Egy megszorítás esetén, ha a mennyiség nem nulla, akkor csak egy megadott időszakra vonatkozhat, mivel akkor a fogyaszthatósággal kapcsolatos döntés csak az időszakban elfogyasztott mennyiséggel felhasználva születhet meg. (Nincs értelme egy konkrét döntésnél időszak és fogyasztás nélkül olyan szabályt felhasználni, ahol a mennyiség nem nulla, mivel ez felhasználás szempontjából értelmezhetetlen.)
- Betegséghez tartozó egészségügyi profil sablon esetén az adott betegség étkezési megkötései szerepelnek a szabályok között.
- Egy szabály típusai, azaz irányai és prioritása:
    - 0 – Tiltás
    - 1 – Nem javasolt
    - 2 – Erősen javasolt
    - 3 – Javasolt
- Idő-mennyiség mátrixot is tárolunk az egészségügyi profil szabályaiban, azaz lehessen megadni egy profilnál nem csak azt, hogy milyen terméket ehet a felhasználó, hanem azt is, hogy az adott megkötés milyen időszakra vonatkozik.
- Mennyiség és időszak megadásának csak együtt van értelme. Mennyiség megadása időszak nélkül értelmezhetetlen, hiszen nem adja meg a fogyasztás időtartományát. Időszak mennyiség nélküli megadása szintén értelmetlen, mivel mennyiség megadása nélkül nincs értelme az időszaki fogyasztás követésének.
- Egy szabály vonatkozhat termék kategóriára, termék elemre és termékre.

Biztosítani kell az egészségügyi szabályok integritását, azaz a következő ellenőrzéseket a szabályok deklarálásakor figyelembe kell venni:

- Nem lehet egy termékre két szabály: Ha egy termékre meghatároztunk egy szabályt, ami után egy másik szabályt is szeretnénk megszabni, akkor a két szabály típusának prioritása meghatározza, hogy melyiket van értelme megtartani, a „gyengébbet” el kell hagynunk. Ha például valaki nem javasolt, hogy túrót egyen, és jön egy új szabály, miszerint tilos neki túrót ennie, akkor az erősebb tiltó szabály alkalmazásának van értelme.

- Ha egy szabály egy termék kategóriára / termék összetevőre / termék elemre vonatkozik, akkor csak erősebb szabályt deklarálhatunk az adott kategóriában lévő termékre / terméket tartalmazó termékekre / termék elemet tartalmazó termékre.

Menü:

- Egy ütemezett, konkrét étrend. Annyival több, mint az egészségügyi profil, hogy itt nem csak megszorításokat teszünk egy adott időszakra, hanem konkrét étrend javaslatot teszünk.

Tehát egy ember étkezései alatt betartandó szabályok három entitásban vannak tárolva. A legalsó szintű az Egészségügyi_profil_szabály. Ez tartalmazza a mennyiségi megkötéseket. Ez egy idő-mennyiség mátrixot is tartalmaz. Ezeket a szabályokat rendeli az emberhez az Egészségügyi_profil entitás. A legfelső megszorításokat tartalmazó entitás a menü, ami étkezések szerinti megkötéseket ír elő. Az időbeli megszorítások bevezetése dinamikus technikákat feltételez.

3. Nem dinamikus algoritmus

Egy konkrét példa egészségügyűi profilra:

- Tiltások: dió > 0g.
- Nem javasolt: energiatartalom > 500 kcal, zsír > 20g, só > 2g.
- Erősen javasolt:üres.
- Javaslat: üres.

Ez a profil azt írja le, hogy diót nem szabad enni, olyan ételt, ami nagyon zsíros (zsír > 20g) vagy sós (só > 2g), azt nem javasolt enni. Megjegyzés, hogy ez a vagy a logikában szokásos megengedő vagy (or), nem a mindennapi nyelvben használt kizáró vagy (xor). A az erősen javasolt és a javaslat ételek listája üres. Látható, hogy a lista elemei közt vagy kapcsolat van. A tiltások általában ételérzékenységekből, allergiákból következnek. A nem javasolt ételekre vonatkozó megszorításokat hosszú távú kockázatok csökkentése miatt írhatja elő az orvos. Nagyon ritkán lehet olyan betegségünk, amikor valamilyen ételt ennünk kell. Ennek két fokozata az erősen javasolt és a javasolt.

A fenti profilt úgy kell értelmezni, hogy a szabályok az étel 100g-jára vonatkoznak. Ha nincs külön kikötve egy szabálynál, akkor az mindig az étel 100g-jára vonatkozik.

Az első szintű szabályellenőrzésnél nincs idő fogalmunk, tehát nem tudjuk azt mondani, hogy napi maximális zsír bevitel nem több 20g-nál, csak azt, hogy minden vizsgált tételnél a zsírtartalom nem több 20g-nál.

Az eFilter rendszerben minden megszorításnál van idő fogalom, kivéve az egyértelmű tiltásokat vagy javaslatokat. Részletesebben, ha a megszorításban van mennyiség megadva, azaz a relációnál megadott mennyiség nagyobb, mint nulla, akkor az mindig valamilyen a szabályban megadott időszakra vonatkozik. Ha nincs mennyiségi korlát, azaz egyértelmű tiltás vagy javaslat van, akkor alkalmazható a lent közölt nem dinamikus algoritmus.

Nézzük az első szintű ellenőrzést végző algoritmust, ami látni fogjuk, hogy egy egyszerű összegzésen alapszik:

3.1. Algoritmus: EHETEM-E

- Input: Étel, Egészségügyi profil
- Output: Tiltott / nem javasolt / javasolt / nem tiltott

1. lépés: Minden az egészségügyi profilban szereplő összetevőhöz készítsünk lokális változókat, kezdő értékük nulla. A fenti példát használva:
   $dió = 0, $energiatartalom = 0, $zsír = 0, $só = 0;
2. lépés: Ha az ételnek megvannak azon adatai, amiket gyűjtünk, akkor hozzáadjuk a változóhoz. Amelyek nincsenek meg a felső szinten, azokat az összetevők alapján számítjuk rekurzívan lefelé haladva, amíg nem találjuk az adott értéket. Ha nincs egyik összetevőben sem a kérdéses érték, akkor feltételezzük, hogy nincs az ételben. A figyelembe vett értéknél mindig 100g-ra kell visszaszámolni.
   Pl.: Vegyük a diós kalács (zsír: 10g, fehérje: 1g, szénhidrát: 96g, liszt: 80g, dió: 10g, vaj: 10g) példáját. Ennek felső szinten nincs megadva az energiatartam és a só, de van dió és zsír. Így a $dió = 10 és a $zsír = 10g számítása készen is van. A többi érték számításához a diós kalács

összetevőinél megadott értékeket kell összeadni. Tegyük fel, hogy ezekből áll a diós kalács: liszt (energiatartalom: 1500kJ, zsír: 0g, szénhidrát:98g, fehérje: 1g), dió (energiatartalom: 800kJ, zsír: 50g, szénhidrát: 0g, fehérje: 50g), vaj (energiatartalom: 2500kJ, zsír: 80g, szénhidrát: 0g, fehérje: 20g). Ebből számítható az energiatartalom:
$energiatartalom = 80/100 * 1500 + 10/100 * 800 + 10/100 * 2500.
Ez a szám úgy jön ki, hogy 100g diós kalácsban 80g liszt van és 100g lisztben 1500 kJ energia, stb… . Mivel só ezeknél sincs megadva, ezért ezeket a termékeket is rekurzívan tovább kell nézni, de csak a só tartalom után kutatva, hiszen a többi értéket már sikerült meghatározni magasabb szinten.

3. lépés: Ha van olyan feltétel a tiltó listán, amely igaz, akkor az algoritmus válasza „tiltott”, egyébként,
4. lépés: Ha van olyan feltétel a nem javasolt listán, amely igaz, akkor az algoritmus válasza „nem javasolt”, egyébként,
5. lépés: Ha javasolt lista minden feltétele igaz és maga a javasolt lista nem üres, akkor az algoritmus válasza „javasolt”, egyébként a válasz „nem tiltott”.

Vegyük észre, hogy a 3-5 lépés egy if – else if szerkezet ágai, ami meghatározza az algoritmus visszatérési értékét. Ahhoz, hogy az étel „javasolt” legyen, ehhez nem lehetett „tiltott” és „nem javasolt”, illetve a „javasolt” lista minden feltételét is ki kell elégítenie az ételhez tartozó értékeknek.

Könnyen észrevehető az algoritmus első javítása. A példában a második lépésben nem érdemes kiszámolni az étel energia tartalmat, hiszen már korábban tudjuk a dió tartalmat, ami pozitív, így kielégíti a dió > 0 tiltó feltételt. Tehát elegendő addig számolni a változók tartalmát, amíg az első tiltó feltétel igazzá nem válik, illetve a tiltó listán kívüli változókat nem érdemes számolni, ha már valamely nem javasolt feltétel igaz lett. Ugyanekkor a tiltott listás változókat tovább kell számolni, hiszen lehet, hogy végül tiltott lesz az étel. A javított algoritmus:

3.2. Algoritmus: EHETEM-E Javított 1.

- Input: Étel, Egészségügyi profil
- Output: Tiltott / nem javasolt / javasolt / nem tiltott

1. lépés: Minden az egészségügyi profilban szereplő összetevőhöz készítsünk lokális változókat, kezdő értékük nulla.
2. lépés: A tiltott listán lévő változók értékét számoljuk. Ha az étel felső szintjén adott az érték, akkor azt másoljuk be a változóba, ha nem, akkor rekurzívan felbontjuk az ételt összetevőire és az ott megadott értékeket (visszaszámolva 100g ételre) adjuk hozzá a változóhoz. Minden rekurzív hívás előtt ellenőrizzük, hogy valamely tiltó feltétel igaz-e. Ha igaz, akkor az algoritmus válasza „tiltott”. Egyébként rekurzívan bontjuk az élelmiszert, amíg kell, vagy amíg lehet.
3. lépés: Hasonlóan kiszámítjuk minden nem javasolt listán lévő változó értékét (feltéve, hogy nem számoltuk már ki a tiltottak közt). Ha valamelyik nem javasolt feltétel igazzá válik a folyamat során, akkor az algoritmus válasza „nem javasolt”.
4. lépés: Hasonlóan kiszámítjuk minden javasolt listán lévő változó értékét (persze csak azokat, amiket még nem számoltunk ki). Miután minden változót kiszámoltunk megnézzük, hogy minden javasolt feltétel igaz-e. Ha igen és a javasolt lista nem üres, akkor az algoritmus válasza „javasolt”.
5. lépés: Az algoritmus válasza „nem tiltott”.

Látható, hogy a 3. lépésre csak akkor jutunk el, ha a második lépésben nem találtunk tiltó feltételt. Hasonló állítás fogalmazható meg a következő lépésekről. Tehát ez az algoritmus lusta abban az értelemben, hogy egy étel valamely értékét (pl. energia tartalom), csak akkor számolja ki, ha az szükséges.

Ugyanakkor a feltételeket intenzíven ellenőrzi, hiszen az étel minden felbontása előtt minden tiltó vagy nem javasolt, stb. feltételt ellenőriz (attól függően, melyik lépésben járunk). Ha sok, bonyolult feltétel van, akkor ez a megoldás rossz!

Egy egyszerű kompromisszum, hogy csak a teljesen kiszámolt értékekre épülő feltételeket értékeljük ki. Ekkor előfordulhat, hogy feleslegesen pontosan számítunk ki valamit, de a feltétel ellenőrzések száma nem az étel összetettségével arányos.

Másik megoldás, hogy a feltételek kiértékelésével bánunk lustán. Érdemes az egy összetevőre vonatkozó több feltételt összevonni és megnézni, hogy ellentmondásos-e. Lehet, hogy eleve ellentmondásos a feltétel rendszer, például: Tiltások: energiatartalom > 2000kJ, energiatartalom < 2500kJ. Azaz nem ehetünk semmit, amiben 2000kJ-nál több az energia; illetve nem ehetünk semmit, amiben nincs legalább 2500kJ energia. Ez a feltétel rendszer önellentmondásos, hiszen vagy az egyik vagy a másik feltétel mindig igaz lesz.

Megjegyzés, eddig feltételeztük, hogy a mennyiségekre vonatkozó feltételek mind nagyobb, esetleg nagyobb egyenlő formájúak, ezért nem volt szükséges pontosan kiszámolni egy értéket. Ha van kisebb vagy kisebb egyenlő alakú feltétel is, akkor pontosan ki kell számolni a változó értékét.

A fenti ötleteket összegzi a következő változata az algoritmusunknak:

3.3. Algoritmus: EHETEM-E Javított 2.

- Input: Étel, Egészségügyi profil
- Output: Tiltott / nem javasolt / javasolt / nem tiltott

1. lépés: Minden az egészségügyi profilban szereplő összetevőhöz készítsünk lokális változókat, kezdő értékük nulla.
2. lépés: A tiltott listán lévő változók értékét számoljuk ki egyenként. Ehhez ezeket a változókat tegyük egy listába. Ezen szaladjunk végig egy ciklussal. A ciklusban:
    a. Vonjuk össze az aktuális változóra vonatkozó feltételeket a tiltó listán. Példa összevonásra: zsír > 100, zsír > 150 összevonva zsír > 100, mert vagy kapcsolat van köztük, így elegendő a leglazább feltételt vizsgálni.
    b. Ha a feltételek ellentettei által meghatározott intervallum üres (azaz önellentmondásos), akkor az algoritmus válasza „tiltott”. Például: energiatartalom > 2000kJ, energiatartalom < 2500kJ.
    c. Számoljuk ki az aktuális változót: Ha az aktuális változó értéke adott az ételnél, akkor azt kell figyelembe venni, ha nem, akkor az étel összetevőnél (illetve azok összetevőinél, rekurzívan lefelé menve, amíg a gyökér nem tartalmazza a kért információt) megadott értékek összege 100g ételre vetítve.
    d. Ha a változó kielégíti az összevont feltételt, akkor az algoritmus válasza „tiltott”.
3. lépés. Hasonlóan dolgozzuk fel a nem javasolt listán lévő feltételek változóit. A ciklus lépései:
    a. Vonjuk össze az aktuális változóra vonatkozó feltételeket a nem javasolt listáról.

b. Ha a feltételek ellentettei által meghatározott intervallum üres (azaz önellentmondásos), akkor az algoritmus válasza „nem javasolt”.
c. Számoljuk ki az aktuális változót: Ha aktuális változó nem nulla, akkor már kiszámításra került, ugrás a 3.d. lépésre. Ha az aktuális változó értéke adott az ételnél, akkor azt kell figyelembe venni, ha nem, akkor az étel összetevőnél megadott értékek összege 100g ételre vetítve.
d. Ha a változó kielégíti az összevont feltételt, akkor az algoritmus válasza „nem javasolt”.

4. lépés. Hasonlóan dolgozzuk fel a javasolt listán lévő feltételek változóit. A ciklus lépései:
   a. Vonjuk össze az aktuális változóra vonatkozó feltételeket a javasolt listáról. Figyelem, itt a változók és kapcsolatban álnak, tehát az összevonásnál a legkeményebb feltételt kell meghagyni.
   b. Ha a feltételek által meghatározott intervallum üres, akkor az algoritmus válasza „nem tiltott”.
   c. Számoljuk ki az aktuális változót: Ha aktuális változó nem nulla, akkor már kiszámításra került, ugrás a 4.d. lépésre. Ha az aktuális változó értéke adott az ételnél, akkor azt kell figyelembe venni, ha nem, akkor az étel összetevőnél megadott értékek összege 100g ételre vetítve.
   d. Ha a változó nem elégíti ki az összevont feltételt, akkor az algoritmus válasza „nem tiltott”.
5. lépés: Az algoritmus válasza „javasolt”.

Vegyük észre, hogy a 4. lépés kicsit változott. Eddig azt ellenőrizte, hogy minden feltétel igaz-e, most azt ellenőrzi, hogy nincs-e hamis feltétel. Ezért változott a 4. és az 5. lépések visszatérési értéke. A 2. lépéstől a 4.a. pontig történik az összevonás. Az összevonás nem feltétlenül eredményez egyszerű feltételt. Ha van nagyobb és kisebb alakú feltétel is, akkor a nagyobb alakúak vonhatók össze egy nagyobb alakúvá, a kisebbek egy kisebb alakúvá.

4. Dinamikus algoritmus

Nézzük azt az esetet, amikor már vagy idő intervallum, vagy étkezés, ami szintén egy időbeni megkötés, is szerepel a feltételek megadásánál.

Egy konkrét példa egészségügyűi profilra:

- Tiltások: dió > 0g
  - napi (0..24 óra): energiatartalom > 30 000kJ.
- Nem javasoltak:
  - napi (0..24 óra): só > 2g
  - este (18..24 óra): zsír > 20g.
- Erősen javasoltak: üres.
- Javaslatok:
  - reggeli (7..9 óra): fehérje > 10g, zsír > 30g.
  - tízórai (9..11 óra): alma = 1db.
  - ebéd (12..14 óra): zsír < 100g.
  - uzsonna (15..17 óra): főtt tojás = 1db.

- vacsora (17..19 óra): zsír < 10g.

Próbáljuk meg értelmezni ezt a profilt. Azt látjuk, hogy diót nem szabad enni, de mikor? Az egyértelmű tiltások és javaslatok mellé nem kell időt írni. Azaz egész nap nem szabad diót enni. Ez időpont hiányát így értelmezzük az eFilter rendszerben. A másik lehetséges használati mód a következő: Ha egy bejegyzés mellett nincs idő megszorítás, akkor azt napi megszorításnak kell venni, tehát az alapértelmezett idő megszorítás a napi.

Azt is látjuk, hogy korlátozva van az összes napi energiatartalom, ami esti étkezéseket tehet tilossá. Nem ajánlott a napi túl sok só és az esti sok zsír. A javaslatok sokkal érdekesebb. Itt egy 5 étkezéses diéta bontakozik ki. Reggelire zsíros falatokkal kell kezdenünk a napot. Tízóraira a rendszer pontosan egy almát ír elő. Tehát semmi mást nem ehetünk. Ebédre csak annyi megkötés van, hogy ne legyen túl zsíros. Uzsonnára pontosan egy főtt tojás engedélyezett. Vacsorára nem túl zsíros koszt van előírva.

Vegyük észre, hogy a tiltó és a nem javasolt listán azt írjuk le, milyen ne legyen az étel, pl.: dió > 0g. Addig a javasolt listán az írjuk le, milyen legyen az étel, azaz mikor ehetem meg a nem tiltott ételeket. Ez azt jelenti, hogy ugyanannak a feltételnek más a jelentése itt és ott. A zsír > 100g jelentése a tiltó listán: Az étel nem tiltott, ha kevesebb vagy pontosan 100g zsírt tartalmaz. A zsír > 100g jelentése a javasolt listán: Az ételnek több, mint 100g zsírt kell tartalmaznia ahhoz, hogy javasolt legyen.

Az étel ellenőrzése a profillal szemben ugyanúgy történik, de figyelembe kell venni, hogy az étel elfogyasztása melyik étkezéshez tartozik. Nyílván elsősorban az ehhez az étkezéshez tartozó szabályokat kell figyelembe venni, de a napi limitek is figyelni kell. A napi limit figyeléséhez tudtom kell az előző étkezések adatait, ami plusz bemeneti információ, illetve nem elég csak azt válaszolnom, hogy egy étel javasolt, hanem a kiszámolt változókat is vissza kell adnom, hogy később újrafelhasználhatóak legyenek.

4.1. Algoritmus: Dinamikus EHETEM-E

- Input: Étel, Egészségügyi profil, étkezés, mai nap elfogyasztott ételek értékei összesítve egy listában.
- Output: Tiltott / nem javasolt / javasolt / nem tiltott, étel értékei egy listában.

1. lépés: Minden az egészségügyi profilban szereplő összetevőhöz készítsünk lokális változókat, kezdő értékük nulla.
2. lépés: Számoljuk ki minden változó értékét és helyezzük el a kimeneti étel értékei listában.
3. lépés: A tiltott lista feltételeit ellenőrizzük. Ehhez a tiltott listán lévő változókat tegyük egy listába. Ezen szaladjunk végig egy ciklussal. A ciklusban:
    a. Vonjuk össze az aktuális változóra vonatkozó feltételeket a tiltó listán. Csak az étkezének megfelelő feltételeket vegyük figyelembe.
    b. Vonjuk össze az aktuális változóra vonatkozó napi feltételeket is a tiltó listán.
    c. Ha a változó kielégíti az összevont étkezési feltételt, akkor az algoritmus válasza „tiltott” és az étel értékeit tartalmazó lista.
    d. Ha a változó plusz a mai nap elfogyasztott ételek értékeit tartalmazó listából ugyanez a változó kielégíti az összevont napi feltételt, akkor az algoritmus válasza „tiltott” és az étel értékeit tartalmazó lista.
4. lépés. Hasonlóan dolgozzuk fel a nem javasolt listán lévő feltételek változóit. A ciklus lépései:

a. Vonjuk össze az aktuális változóra vonatkozó feltételeket a nem javasolt listán. Csak a étkezének megfelelő feltételeket vegyük figyelembe.
b. Vonjuk össze az aktuális változóra vonatkozó napi feltételeket is a nem javasolt listán.
c. Ha a változó kielégíti az összevont étkezési feltételt, akkor az algoritmus válasza „nem javasolt” és az étel értékeit tartalmazó lista.
d. Ha a változó plusz a mai nap elfogyasztott ételek értékeit tartalmazó listából ugyanez a változó kielégíti az összevont napi feltételt, akkor az algoritmus válasza „nem javasolt” és az étel értékeit tartalmazó lista.

5. lépés. Hasonlóan dolgozzuk fel a javasolt listán lévő feltételek változóit. A ciklus lépései:
a. Vonjuk össze az aktuális változóra vonatkozó feltételeket a javasolt listán. Csak a étkezének megfelelő feltételeket vegyük figyelembe.
b. Vonjuk össze az aktuális változóra vonatkozó napi feltételeket is a javasolt listán.
c. Ha a változó nem elégíti ki az összevont étkezési feltételt, akkor az algoritmus válasza „nem tiltott” és az étel értékeit tartalmazó lista.
d. Ha a változó plusz a mai nap elfogyasztott ételek értékeit tartalmazó listából ugyanez a változó nem elégíti ki az összevont napi feltételt, akkor az algoritmus válasza „nem tiltott” és az étel értékeit tartalmazó lista.
6. lépés: Az algoritmus válasza „javasolt” és az étel értékeit tartalmazó lista.

Vegyük észre, hogy ez az algoritmus nagyon mohó. Minden változót kiszámol, akkor is, ha tiltott az étel. Ez azért van, mert vissza kell adnunk az étel értékeit. Ugyanakkor egy tiltott ételnél ez felesleges, hiszen valószínűleg nem eszi meg a felhasználó. Igaz, ebben nem lehetünk biztosak. Sőt, lehet, hogy a javasolt ételt sem fogja megenni és feleslegesen számoljuk ki minden értékét. A felesleges számolások elhagyására a következő módon javítjuk az algoritmust. Ha kiszámoltunk egy értékét az ételnek, ami az étel felső szintjén nincs rögzítve, akkor ezt rögzítjük az étel felső szintjén, ami ugyan növeli az adatbázist, sőt redundánssá teszi, de a számolás vele sokkal gyorsabb lesz. Kellene adnunk egy kiegészítő algoritmust is, amit akkor kell meghívni, ha tényleg megettük az ételt (akár tiltott volt, akár nem). Ez használná a legfelső szintre írt értékeket a mai nap elfogyasztott ételek étékeinek kiszámításához. Ugyanakkor ez az algoritmus triviális, így nem közöljük.

4.2. Algoritmus: Dinamikus EHETEM-E Javított 1.

- Input: Étel, Egészségügyi profil, étkezés, mai nap elfogyasztott ételek értékei összesítve egy listában.
- Output: Tiltott / nem javasolt / javasolt / nem tiltott.

1. lépés: Minden az egészségügyi profilban szereplő összetevőhöz készítsünk lokális változókat, kezdő értékük nulla.
2. lépés: A tiltott listán lévő változók értékét számoljuk ki egyenként. Ehhez ezeket a változókat tegyük egy listába. Ezen szaladjunk végig egy ciklussal. A ciklusban:
a. Vonjuk össze az aktuális változóra vonatkozó feltételeket a tiltó listán. Csak a étkezének megfelelő feltételeket vegyük figyelembe.
b. Vonjuk össze az aktuális változóra vonatkozó napi feltételeket is a tiltó listán.
c. Számoljuk ki az aktuális változót. Ha ehhez rekurzívan fel kellett dolgozni az ételt, akkor az adatbázisban az étel leírását egészítsük ki ezzel az értékkel.
d. Ha a változó kielégíti az összevont étkezési feltételt, akkor az algoritmus válasza „tiltott”.

    e. Ha a változó plusz a mai nap elfogyasztott ételek értékeit tartalmazó listából ugyanez a változó kielégíti az összevont napi feltételt, akkor az algoritmus válasza „tiltott”.
3. lépés. Hasonlóan dolgozzuk fel a nem javasolt listán lévő feltételek változóit. A ciklus lépései:
    a. Vonjuk össze az aktuális változóra vonatkozó feltételeket a nem javasolt listán. Csak a étkezének megfelelő feltételeket vegyük figyelembe.
    b. Vonjuk össze az aktuális változóra vonatkozó napi feltételeket is a nem javasolt listán.
    c. Számoljuk ki az aktuális változót. Ha ehhez rekurzívan fel kellett dolgozni az ételt, akkor az adatbázisban az étel leírását egészítsük ki ezzel az értékkel.
    d. Ha a változó kielégíti az összevont étkezési feltételt, akkor az algoritmus válasza „nem javasolt”.
    e. Ha a változó plusz a mai nap elfogyasztott ételek értékeit tartalmazó listából ugyanez a változó kielégíti az összevont napi feltételt, akkor az algoritmus válasza „nem javasolt”.
4. lépés. Hasonlóan dolgozzuk fel a javasolt listán lévő feltételek változóit. A ciklus lépései:
    a. Vonjuk össze az aktuális változóra vonatkozó feltételeket a javasolt listán. Csak a étkezének megfelelő feltételeket vegyük figyelembe.
    b. Vonjuk össze az aktuális változóra vonatkozó napi feltételeket is a javasolt listán.
    c. Számoljuk ki az aktuális változót. Ha ehhez rekurzívan fel kellett dolgozni az ételt, akkor az adatbázisban az étel leírását egészítsük ki ezzel az értékkel.
    d. Ha a változó nem elégíti ki az összevont étkezési feltételt, akkor az algoritmus válasza „nem tiltott”.
    e. Ha a változó plusz a mai nap elfogyasztott ételek értékeit tartalmazó listából ugyanez a változó nem elégíti ki az összevont napi feltételt, akkor az algoritmus válasza „nem tiltott”.
5. lépés: Az algoritmus válasza „javasolt”.

Lehet látni, hogy az inputban megadott étkezést, tehát hogy mikor szeretnénk az ételt elfogyasztani, több helyen, nevezetesen a 2-4.a. pontban használjuk fel. A listákról azokat a feltételeket vonjuk össze, amik ehhez az étkezéshez tartoznak.

Az algoritmusnál az okozhat még gondot, hogy nap felosztása az étkezések között többféleképpen is lehetséges. Például (zárójelben a körülbelüli időpont):

- Reggeli (7 óra), tízórai (10 óra), ebéd (13 óra), uzsonna (16 óra), vacsora (19 óra), azaz napi 5 étkezés.
- Reggeli (8 óra), ebéd (12 óra), uzsonna (16 óra), vacsora (20 óra), azaz napi 4 étkezés.
- Reggeli (8 óra), ebéd (13 óra), vacsora (18 óra), azaz napi 3 étkezés.
- Villás reggeli (11 óra), ebéd (17 óra), vacsora (23 óra).

Látható, hogy nem egyértelmű, hogy ebéd alatt mit érthet a szabály megfogalmazója. Pedig igazán extrém étkezési szokásokat nem is soroltunk fel. Több megoldás is kínálkozok a probléma megoldására:

- Étkezés helyett sorszám megadása, azaz, hogy hányadik étkezéshez tartozik a szabály.
- Étkezés helyett időpont megadása a szabálynál. Egy étkezésre két szabály közül az vonatkozik, amelyik közelebbi időpontot tartalmaz.

- Étkezés helyett időintervallum megadása a szabálynál. Egy étkezésre vonatkozik a szabály, ha az adott időintervallumba esik, egyébként nem.

Az első megoldás egyértelműen rossz, mert ha kimarad egy étkezés, akkor azt jelezni kellene a rendszernek, ami feleslegesen bonyolítaná a felhasználói felületet. A második megoldás jónak tűnik. Megadjuk, hogy ideálisan mikor egyen a felhasználó. Ha ettől nem tér el túl sokkal, akkor azt a rendszer helyesen kezeli. A harmadik megoldásnál gond lehet, hogy mit kezdjek egy hajnali túró rúdival, vagy egy késő esti chipsszel. A második megoldás ezt a helyén kezelné, a túró rúdit a reggelinél, a chipset a vacsoránál. Így egyszerűsíthető az algoritmus inputja, hiszen nem kell megadni, milyen étkezésről van szó. Ez egyszerűsíti a felhasználói felületet is, hiszen a felhasználónak nem kell megmondania, mihez (reggeli / ebéd / stb…) akarja megenni az ételt. Az így megváltoztatott algoritmus:

4.3. Algoritmus: Dinamikus EHETEM-E Javított 2.

- Input: Étel, Egészségügyi profil, mai nap elfogyasztott ételek értékei összesítve egy listában.
- Output: Tiltott / nem javasolt / javasolt / nem tiltott.

1. lépés: Az $idő változóba elmentjük a pontos időt.
2. lépés: Minden az egészségügyi profilban szereplő összetevőhöz készítsünk lokális változókat, kezdő értékük nulla.
3. lépés: A tiltott listán lévő változók értékét számoljuk ki egyenként. Ehhez ezeket a változókat tegyük egy listába. Ezen szaladjunk végig egy ciklussal. A ciklusban:
    a. Vonjuk össze az aktuális változóra vonatkozó feltételeket a tiltó listán. Csak a közeli szabályokat, azaz az $idő-$delta és a $idő+$delta közt lévő szabályokat vesszük figyelembe. A $delta számításához az összes szabályt, kivéve a napi szabályokat, sorba állítjuk. A sorból ciklikus sort készítünk. A ciklikus sorban megkeressük az $idő helyét. A $kisebb változóba mentjük az $idő-nél kisebb, a $nagyobb változóba a nagyobb időpontot. $delta = min(ads($idő-$kisebb), abs($nagyobb-$idő)). Ha $delta kisebb, mint 1 óra, akkor $delta legyen 1 óra. Az ezen szabályok összevonásával kapjuk az úgynevezett összevont közeli feltételt.
    b. Vonjuk össze az aktuális változóra vonatkozó napi feltételeket is a tiltó listán.
    c. Számoljuk ki az aktuális változót. Ha ehhez rekurzívan fel kellett dolgozni az ételt, akkor az adatbázisban az étel leírását egészítsük ki ezzel az értékkel.
    d. Ha a változó kielégíti az összevont közeli feltételt, akkor az algoritmus válasza „tiltott”.
    e. Ha a változó plusz a mai nap elfogyasztott ételek értékeit tartalmazó listából ugyanez a változó kielégíti az összevont napi feltételt, akkor az algoritmus válasza „tiltott”.
4. lépés. Hasonlóan dolgozzuk fel a nem javasolt listán lévő feltételek változóit. A ciklus lépései:
    a. Vonjuk össze az aktuális változóra vonatkozó feltételeket a nem javasolt listán. Csak a közeli feltételeket vegyük figyelembe.
    b. Vonjuk össze az aktuális változóra vonatkozó napi feltételeket is a nem javasolt listán.
    c. Számoljuk ki az aktuális változót. Ha ehhez rekurzívan fel kellett dolgozni az ételt, akkor az adatbázisban az étel leírását egészítsük ki ezzel az értékkel.
    d. Ha a változó kielégíti az összevont közeli feltételt, akkor az algoritmus válasza „nem javasolt”.

e. Ha a változó plusz a mai nap elfogyasztott ételek értékeit tartalmazó listából ugyanez a változó kielégíti az összevont napi feltételt, akkor az algoritmus válasza „nem javasolt”.

5. lépés. Hasonlóan dolgozzuk fel a javasolt listán lévő feltételek változóit. A ciklus lépései:
   a. Vonjuk össze az aktuális változóra vonatkozó feltételeket a javasolt listán. Csak a közeli feltételeket vegyük figyelembe.
   b. Vonjuk össze az aktuális változóra vonatkozó napi feltételeket is a javasolt listán.
   c. Számoljuk ki az aktuális változót. Ha ehhez rekurzívan fel kellett dolgozni az ételt, akkor az adatbázisban az étel leírását egészítsük ki ezzel az értékkel.
   d. Ha a változó nem elégíti ki az összevont közeli feltételt, akkor az algoritmus válasza „nem tiltott”.
   e. Ha a változó plusz a mai nap elfogyasztott ételek értékeit tartalmazó listából ugyanez a változó nem elégíti ki az összevont napi feltételt, akkor az algoritmus válasza „nem tiltott”.
6. lépés: Az algoritmus válasza „javasolt”.

A közeli feltétel fogalmát több helyen is használjuk, de csak a 3.a. pontban magyarázzuk el. Ugyanott adjuk meg az összevont közeli feltétel fogalmát is.

A $delta számításánál figyelembe lehet venni az alvást, ami az étkezések területén is nagy vízválasztó. Erre azért lehet szükség, mert egy hajnal 2 órai evés inkább a vacsorához (19 óra) számítandó, mint a reggelihez (7 óra), habár a reggelihez közelebbi időpontban van, de még az alvás előtt. Ezzel tovább lehet finomítani a fenti algoritmust.

5. Szűrése indexeléssel

Adatbázisok esetén bevett gyakorlat, hogy indexek segítségével gyorsítjuk a lekérdezéseket. Azokra az oszlopokra, amelyekre feltételek vonatkoznak, azokra érdemes indexet tenni, mert ez néha nagyságrendekkel gyorsíthatja a keresést. Ugyanakkor az indexek tárigényesek, illetve karbantartást igényelnek, de ezt a mai adatbázis-kezelők elrejtik előlünk.

Természetes ötlet, hogy az egészségügyi profil alapján végzett élelmiszer szűréshez is használjuk fel az indexeket. Ebben a fejezetben nem feltételezzük adatbázis használatát, de leírtak alapján könnyen létrehozható adatbázis-kezelőkben is a megfelelő indexek.

Az ilyen fajta indexelést a szakirodalom bitmap indexnek nevezi.

5.1. Elméleti háttér

Az adatkezelő rendszerek egyik hatékony indexelési technikája a bitmap indexek alkalmazása. A bitmap index struktúrát 1985-ben vezették be [SM1985] az összetett feltételeknek megfelelő rekordok hatékony keresése céljából, néhány javítás után [CI1998] széles körben elterjedt. Az alap bitmap index egy bittérképet jelent, ahol a kulcs minden értékének megfelel egy sor a mátrixban és minden rekord egy oszlophoz tartozik. A 1-es érték jelenti, hogy a rekord megfelel a sort jelentő feltételnek. A feltétel lehet egyezőség a kulccsal vagy az, hogy nagyobb, mint a kulcs (tartomány bitmap). A bitmap index előnyei:

- kis értéktartomány esetén tömör a tárolás,

- hatékonyság az összetett feltétetek kezelése esetén (bitműveletekre visszavezethető feltételek esetén).

Egy K méretű értéktartományú és N rekordot tartalmazó minta esetén a bitmap indexben történő keresés blokk-költsége:

O(N/m)

ahol m az egy blokk tárolási egységen jegyzett elemek darabszáma. Ha egy klasszikus B-fa indexet veszünk, akkor ott

O(log(kN))

költséget kapunk. Ha m kicsi, pl. 0.001, akkor kb. 20000 rekordnál már a B-fa index hatékonyabban működne, hiszen a bitmap index mérete is lineárisan nő. A méret okozta probléma megoldáséra több módszer is kidolgozásra került. A legelterjedtebb megoldások:

- a bitmap tömörítése, például a WAH [WOS2006] módszer RLE kódolást használ,
- a kulcstartomány reprezentálás csökkentése (a kulcsérték közvetlen ábrázolása helyett a komponenseit ábrázolják).

Mivel a teljes bitmap mérete és a keresés is függ a kulcsok darabszámától, a bitmap index csak kis értékkészletű kulcsoknál célszerű használni. Egy másik szempontból pedig azt is figyelembe kell venni, hogy az indexen keresztüli elérés többlet költséget jelent a közvetlen SCAN művelettel szemben. Ezért kis méreteknél a SCAN eljárás hatékonyabb. A bitmap indexek fő ereje akkor jelenik meg, amikor

- nagy rekordszám, de kevés különböző kulcs,
- egy mezőre több feltétel is megjelenik.

Az Oracle RDBMS rendszerek implementált bitmap index (Oracle Bitmap Index Techniques: http://www.dba-oracle.com/ oracle_tips_bitmapped_indexes.htm) is tömörített formában tárolja az index adatokat. A mérési eredmények alapján a keresési költség lineárisan arányos a kulcsok darabszámával. Kis táblaméreteknél a CBO optimalizáló motor a SCAN módszert alkalmazza.

Az adatok elérésének hatékonyság növelésében egy további gyakran alkalmazott eszköz az előszámítások elvégzése. Az adattárház rendszerek [In2005] egyik fő jellemzője, hogy a különböző szintű aggregált adatokat előre meghatározza és letárolja azok értékét. Az aggergált adatokra vonatkozó későbbi lekérdezés során tehát nem kell már számításokat végezni, a letárolt értékeket adja vissza a rendszer. Az előaggregáció elvégzésére több algoritmus is kidolgozásra került, az adatkockák esetében a greedy-algoritmus [LXLQ2009] az egyik legismertebb változat.

Az feladatunkban előaggregációra nyílik lehetőség az egyes ételeknél a különböző alapanyagok hiányzó tartalmazási értékének nyilvántartásánál. Az ételek hierarchikus felépítése miatt az alacsonyabb szintű összetevők relatív mennyisége alapesetben csak több iterációs számítás után határozható meg. Mivel ilyenkor a teljes összetevő adatbázisból kell ismételten adatokat keresni, célszerű ezen számításokat megspórolni. Az előszámítások révén minden bekerülő új elemnél a letároláskor meghatározásra kerülnek az elemi szintű komponensek mennyiségei és a kiszámított értékek bekerülnek a komponens leíró táblába.

Az előszámítások elvégzése az adatbázison belül egy trigger mechanizmushoz kötődhet:

CREATE TRIGGER szamitas AFTER INSERT OR UPDATE ON tábla FOR EACH ROW rutin

Az elvégzett előszámítások révén csökkenthető a lekérdezés alatti művelet végrehajtás költsége.

5.2. Indexelés

A fejezet alap ötlete az, hogy egy mennyiségi megszorítás ellenőrzését vezessük vissza egyszerű ellenőrzésre, hogy az elfogyasztandó étel indexelt-e vagy sem.

Vegyünk egy egyszerű megszorítást: 30g >= fehérje > 10g, azaz legalább 10g+epszilon fehérjét ennünk kell, de legfeljebb 30g-ot. Tegyük fel, hogy létezik 4 indexünk minden az adatbázisban meglévő élelmiszerhez. Egy élelmiszer indexelt a 0. index által, ha egyáltalán nem tartalmaz fehérjét. Az 1. index a 10g-nál kevesebb, vagy pont 10g fehérjét tartalmazó ételeket indexelje. A 2. index <=20g fehérje tartalomnak megfelelőeket indexeli. A 3. index a <=30g feltétel alapján indexel. Természetesen nem csak a <= bináris összehasonlító művelet alapján lehet indexelni, hanem bármilyen bináris összehasonlító művelettel. A különböző indexek megkülönböztetésére az index első helyére a használt összehasonlító műveletet írjuk.

Az az étel, amiben 12g fehérje van, az (<=,0,0,1,1) index négyessel bír, mert a 0. és az 1. index nem a 2. és a 3. index pedig indexeli. Ebben az esetben a fenti feltétel (30g >= fehérje > 10g) megfelel annak az ellenőrzésnek, hogy a 2. és a 3. index indexeli, a 1. index nem, azaz a (<=,x,0,1,1) négyessel bír, ahol x értéke mindegy (habár a monotonitás miatt kitalálható). Látható, hogy a megszorítást ellenőrzést így maximum 4 index vizsgálatra tudjuk redukálni.

A fenti példában bemutatott (<=,x,0,1,1) index n-est a továbbiakban **maszk**nak nevezzük. Nézzük néhány fehérje tartalom maszkját:

- 0 fehérje tartalom maszkja: (<=,1,1,1,1)
- 5 fehérje tartalom maszkja: (<=,0,1,1,1)
- 10 fehérje tartalom maszkja: (<=,0,1,1,1)
- 15 fehérje tartalom maszkja: (<=,0,0,1,1)
- 20 fehérje tartalom maszkja: (<=,0,0,1,1)
- 25 fehérje tartalom maszkja: (<=,0,0,0,1)
- 30 fehérje tartalom maszkja: (<=,0,0,0,1)
- 35 fehérje tartalom maszkja: (<=,0,0,0,0)
- 40 fehérje tartalom maszkja: (<=,0,0,0,0)

A fenti példában a következő maszkok lehetségesek, amelyek a következő megszorításokat kódolják:

- (<=,1,1,1,1) = (<=,1,x,x,x) – fehérje tartalom <= 0g, azaz, fehérje tartalom = 0g, azaz, nyomokban sem tartalmaz fehérjét.
- (<=,x,1,1,1) = (<=,x,1,x,x) – fehérje tartalom <= 10g.
- (<=,x,x,1,1) = (<=,x,x,1,x) – fehérje tartalom <= 20g.
- (<=,x,x,x,1) = (<=,x,x,x,1) – fehérje tartalom <= 30g.
- (<=,0,x,x,x) = (<=,0,x,x,x) – fehérje tartalom > 0g.
- (<=,0,0,x,x) = (<=,x,0,x,x) – fehérje tartalom > 10g.

- (<=,0,0,0,x) = (<=,x,x,0,x) – fehérje tartalom > 20g.
- (<=,0,0,0,0) = (<=,x,x,x,0) – fehérje tartalom > 30g.
- (<=,0,1,1,1) = (<=,0,1,x,x) – 0g < fehérje tartalom <= 10g.
- (<=,0,x,1,1) = (<=,0,x,1,x) – 0g < fehérje tartalom <= 20g.
- (<=,0,x,x,1) = (<=,0,x,x,1) – 0g < fehérje tartalom <= 30g.
- (<=,0,0,1,1) = (<=,x,0,1,x) – 10g < fehérje tartalom <= 20g.
- (<=,0,0,x,1) = (<=,x,0,x,1) – 10g < fehérje tartalom <= 30g.
- (<=,0,0,0,1) = (<=,x,x,0,1) – 20g < fehérje tartalom <= 30g.

Ebben a cikkben **pozitív index**nek nevezzük a <= összehasonlítást használó indexeket és **negatív index**nek a > összehasonlítást használókat. Könnyen belátható, hogy a (<=,1,1,1,1) és a (>,0,0,0,0) maszk ugyanazt a megszorítást kódolja. Hasonlóan egyszerű belátni általánosan is, hogy a bitek és a összehasonlítás megfordításával a pozitív maszkból azt a negatív maszkot kapjuk, ami ugyanazt a megszorítást kódolja. Ez fordítva is igaz.

Ha egy maszkkal pontosan tudunk leírni egy megszorítást, az inkább véletlen, mert sokkal több olyan eset van, amikor ez nem lehetséges. Pl. a 28g >= fehérje > 25g feltételt csak közelíteni tudjuk a (<=,x,x,0,1) maszkkal, ami a 30g >= fehérje > 20g feltételnek felel meg. Ez a megszorítás tartalmazza az eredetit, ezért **tartalmazó megszorítás**nak nevezzük.

Ez alapján a **legkisebb tartalmazó maszk (LKTM)** egy olyan maszk, ami tartalmazó megszorítást ír le és nincs olyan maszk, amelyhez nála kisebb intervallumot és tartalmazó megszorítást írna le. Könnyen belátható, hogy minden „M >= összetevő tartalom > N” alakú megszorításhoz, amelynél M (azaz a felső határ) kisebb, mint a legnagyobb indexelt érték, adható legkisebb tartalmazó maszk.

Hasonlóan értelmezhető a **legnagyobb tartalmazott maszk (LNTM)** is, de az csak akkor adható meg, ha az eredeti megszorítás felső határánál létezik kisebb indexelt érték, az alsó határnál pedig nagyobb és a kettő nem ugyan az. Így például a 28g >= fehérje > 25g megszorításhoz nem létezik legnagyobb tartalmazott maszk. A következő példánál viszont létezik: A 38g >= fehérje > 15g megszorításhoz tartozó legnagyobb tartalmazott maszk a (<=,x,x,0,1), ami a 30g >= fehérje > 20g megszorítást kódolja. Ha a megszorítás „összetevő tartalom > N” alakú vagy „M >= összetevő tartalom” alakú, és M illetve N kisebb, mint a legnagyobb indexelt érték, akkor mindig megadható a legnagyobb tartalmazott maszk.

A legkisebb tartalmazó maszk az eredetinél megengedőbb vagy egyenlő, azaz több vagy ugyanannyi ételre igaz. A legnagyobb tartalmazott maszk az eredetinél szigorúbb vagy egyenlő megszorítást ír le, azaz kevesebb vagy ugyanannyi ételre igaz.

5.3. Indexelés logaritmikus index sűrűséggel

A fenti példákban 10g-os lépésközzel indexeltük az élelmiszer fehérje tartalmát. Már csak az a kérdés, milyen sűrűn legyenek az indexek, hogy az eredetei megszorításnál csak kicsit megengedőbb, vagy csak kicsit szigorúbb megszorításokat tudjunk velük ellenőrizni, ugyanakkor ne legyen túl sok index se.

Jelen cikkben az a javaslatunk, hogy minden kettő hatványra tegyünk indexet. A mértékegység összetevőnként lehet más és más. A fehérjénél ez lehet gramm, de mogyorónál, ahol már kis

mennyiség is túlzott reakciót válthat ki, a mértékegység lehet milligramm. Tehát a következő pozitív vagy negatív indexek kerülnek ki minden összetevőre (a mértékegység helyette egység szerepel):

- 0. index: Az élelmiszer indexelt, ha az összetevő tartalma nagyobb, mint 0 egység.
- 1. index: Az élelmiszer indexelt, ha az összetevő tartalma nagyobb, mint 1 egység.
- 2. index: Az élelmiszer indexelt, ha az összetevő tartalma nagyobb, mint 2 egység.
- …
- n. index: Az élelmiszer indexelt, ha az összetevő tartalma nagyobb, mint 2^(n-1) egység.

Ha az egység milligramm, akkor a 18. index 0,131072 kg-nál nagyobb összetevő tartalmat indexel. Ennek az indexelésnek az az előnye, hogy milligrammos tartományban nagyon pontosan, ahol ez el is várt. Ahol pedig nem szükséges a nagy pontosság, azaz a kg-os kategóriában, ott csak hozzávetőleges.

Sok összetevőre valószínűleg nem kell 20 index, hiszen, ha pozitív indexelés esetén az index feltételének egy élelmiszer sem felel meg, akkor az az index felesleges. Illetve negatív indexelés esetén, ha minden élelmiszer megfelel neki, akkor is felesleges.

Nézzük néhány szám pozitív és negatív maszkját:

- 0 negatív és pozitív maszkja: (>,0,0,0,0); (<=,1,1,1,1)
- 1 negatív és pozitív maszkja: (>,1,0,0,0); (<=,0,1,1,1)
- 2 negatív és pozitív maszkja: (>,1,1,0,0); (<=,0,0,1,1)
- 3 negatív és pozitív maszkja: (>,1,1,1,0); (<=,0,0,0,1)
- 4 negatív és pozitív maszkja: (>,1,1,1,0); (<=,0,0,0,1)
- 5 negatív és pozitív maszkja: (>,1,1,1,1); (<=,0,0,0,0)
- 6 negatív és pozitív maszkja: (>,1,1,1,1); (<=,0,0,0,0)

A következő példában a négyzetes lépésközt és negatív indexelést használva adjuk meg a 4 széles maszkok jelentését, feltéve, hogy az index a fehérjetartalmat indexeli:

- (>,0,0,0,0) = (>,0,x,x,x) = (<=,1,x,x,x) – fehérje tartalom = 0 egység.
- (>,x,0,0,0) = (>,x,0,x,x) = (<=,x,1,x,x) – fehérje tartalom <= 1 egység.
- (>,x,x,0,0) = (>,x,x,0,x) = (<=,x,x,1,x) – fehérje tartalom <= 2 egység.
- (>,x,x,x,0) = (>,x,x,x,0) = (<=,x,x,x,1) – fehérje tartalom <= 4 egység.
- (>,1,x,x,x) = (>,1,x,x,x) = (<=,0,x,x,x) – fehérje tartalom > 0 egység.
- (>,1,1,x,x) = (>,x,1,x,x) = (<=,x,0,x,x) – fehérje tartalom > 1 egység.
- (>,1,1,1,x) = (>,x,x,1,x) = (<=,x,x,0,x) – fehérje tartalom > 2 egység.
- (>,1,1,1,1) = (>,x,x,x,1) = (<=,x,x,x,0) – fehérje tartalom > 4 egység.
- (>,1,0,0,0) = (>,1,0,x,x) = (<=,0,1,x,x) – 0 egység < fehérje tartalom <= 1 egység.
- (>,1,x,0,0) = (>,1,x,0,x) = (<=,0,x,1,x) – 0 egység < fehérje tartalom <= 2 egység.
- (>,1,x,x,0) = (>,1,x,x,0) = (<=,0,x,x,1) – 0 egység < fehérje tartalom <= 4 egység.
- (>,1,1,0,0) = (>,x,1,0,x) = (<=,x,0,1,x) – 1 egység < fehérje tartalom <= 2 egység.
- (>,1,1,x,0) = (>,x,1,x,0) = (<=,x,0,x,1) – 1 egység < fehérje tartalom <= 4 egység.
- (>,1,1,1,0) = (>,x,x,1,0) = (<=,x,x,0,1) – 2 egység < fehérje tartalom <= 4 egység.

A fenti táblázatban megadtuk mind a negatív, mind a pozitív maszkot, ami leírja a megszorítást. Látható, hogy mindig két értéket kell csak vizsgálni a maszkból. A legelső negatív index maszk azt

mutatja meg, hogy mi az x jelentése a második negatív alakban, azaz, a második negatív alakban lévő 1 előtt csak 1 állhat, illetve a 0 után csak 0, a kettő közt állhat 0 és 1 is.

Elképzelhető, hogy technikai megszorítások miatt csak azt tudjuk ellenőrizni, hogy a maszkban van-e 1, azt nem, hogy van-e 0. Ilyen esetben használni kell a negatív és a pozitív indexelést is. Mivel a kettőben a bitek pont átfordított helyzetben vannak, mint a másikban, ezért például a negatív maszkban lévő 0 meglétét úgy tudom ellenőrizni, hogy megnézem, hogy ugyanazon a helyen a pozitív maszkban 1 van-e. Hasonlóan ellenőrzöm a pozitívban lévő 0 meglétét a negatív index segítségével.

A fenti trükköt kell alkalmaznunk akkor, ha a maszkot egy 32 bites (vagy ahány bites processzorral dolgozunk) bit sorozatként, azaz egy gépi szó típusú számként, tároljuk. Ilyenkor az K.-dik helyen lévő 1 meglétét úgy ellenőrizzük, hogy a szám >= 2^(K+1)-1 feltétel igaz-e, feltéve, hogy a számban a bitek sorrendje épp fordított, mint a maszkban, illetve negatív indexelést használunk. A Tesztelés során pontosan ezt a megközelítést használtuk, annyi különbséggel, hogy nem fordítottuk meg a bitek sorrendjét, ami így utólag nézve okozott néhány nehézséget, de alapvetően nem befolyásolta a teszt hatékonyságát.

5.4. Algoritmus: Index Maszkos EHETEM-E (IMEE)

A fentieket figyelembe véve a következő algoritmus adható, amely már feltételezi, hogy a kettő hatványaira elhelyeztük a pozitív indexeket. Illetve az egészségügyi profilban lévő megszorítás mértékegységei megegyezik az indexelt érték mértékegységével minden összetevő esetén. Illetve minden megszorítás felső korlátja kisebb, mint a legnagyobb indexelt érték.

- Input: Egészségügyi profil, Élelmiszerek adatbázisa minden összetevő kettő hatványára indexelve.
- Output: Tiltott élelmiszerek listája.

1. lépés: Az egészségügyi profil tiltó megszorításait összetevőnként összevonom „N egység < összetevő tartalom <= M egység” alakú megszorítássá. Ez feltételezi, hogy minden megszorítás „X egység < összetevő tartalom”, „összetevő tartalom <= Y egység” vagy „X egység < összetevő tartalom <= Y egység” alakú. Ez a feltevés az eFilter projektben elfogadott.
2. lépés: Az ilyen fajta megszorítást pozitív maszkkal könnyen leírhatjuk. Ehhez először határozzuk meg az L és a H értéket. Mint látni fogjuk az L a 0, a H az 1 helyét határozza meg a „N egység < összetevő tartalom <= M egység” megszorításhoz tartozó maszkban.
    a. Ha N = 0, akkor L = 0, egyébként L = padlás($\log_2$(N))+1, ahol padlás(x) felfelé kerekít. Ha N nem létezik, azaz az összevont megszorítás „összetevő tartalom <= M egység” alakú, akkor L se létezik.
    b. Ha M = 0, akkor H = 0, egyébként H = padló($\log_2$(M))+1, ahol padló(x) lefelé kerekít. Ha M nem létezik, azaz az összevont megszorítás „ N egység < összetevő tartalom” alakú, akkor H se létezik.
3. lépés: Minden összetevőre kiszámolom az L és a H értéket az összevont feltétel alapján. Az így kapott L és H értékre szűkítem az adatbázist, úgy hogy csak azok az ételeket maradjanak, ahol az összetevő maszkja (<=,x,…,x,0,x,…x,1,x,…,x), ahol
    a. a 0 az L-dik indexen áll, illetve nincs 0 a maszkban, ha L nem létezik.
    b. az 1 a H-dik indexen áll, illetve nincs 1 a maszkban, ha H nem létezik.
    Megjegyzés: A maszkban 0-tól kezdődik az indexelés.

4. lépés: Vegyük észre, hogy ez az összevont feltétel legnagyobb tartalmazott maszkja (LNTM). Ha a maszkban a 0 és az 1 indexe egy helyre esik, akkor nem létezik a legnagyobb tartalmazott maszk. Ebben az esetben az algoritmus üres listát ad vissza.
5. lépés: A megmaradt ételek az összes összevont feltételnek megfelelnek, ezeket visszaadom.

A fenti algoritmust röviden IMEE algoritmusnak neveztük el. A legnagyobb gond vele, hogy nem mindig létezik a legnagyobb tartalmazott maszk. Szerencsére a gyakorlatban a tiltó megszorítások majdnem mindig „összetevő tartalom > X egység" alakúak. Ilyenkor mindig van legnagyobb tartalmazott maszk is, feltéve, hogy elég nagy a legnagyobb indexelt érték.

Az IMEE algoritmus elviekben más, mint a korábbiak, hiszen nem egy ételre kérdez rá, hogy az tiltott vagy sem, hanem étel listát ad vissza. A fenti verziójában a tiltott ételek listáját. Az IMEE algoritmussal nem csak a tiltott, hanem a tiltott / nem javasolt / erősen javasolt / javasolt élelmiszerek listája is visszaadható. A nem tiltott élelmiszerek listája megkapható a tiltott élelmiszerek összes élelmiszerből történő kivonásával, vagy ha negáljuk a tiltó feltételeket és az alapján szűrünk.

Nézzünk egy példát az IMEE algoritmus futására. Tegyük fel, hogy a tiltásaink a következőek:

- zsír > 3dkg, zsír > 2dkg
- fehérje > 1,5dkg, fehérje > 8 dkg
- dió > 0g

Ezek az összevonások után így néznek ki:

- zsír > 2dkg
- fehérje > 1,5dkg
- dió > 0g

Az ezekhez tartozó legnagyobb tartalmazott maszkok:

- (<=,x,x,0,x) – zsír tartalom > 2 dkg.
- (<=,x,x,0,x) – fehérje tartalom > 2 dkg.
- (<=,0,x,x,x) – dió tartalom > 0 g.

Végül az algoritmus azokat az ételeket adja vissza, amelyek mind a három maszknak megfelelnek. Az első és a harmadik maszk pontosan írja le a megszorítást, a második egy erősebb megszorításnak felel meg, mint az eredeti. Így nem minden nem tiltott élelmiszert ad vissza, de amit visszaad, az biztosan nem tiltott.

Az IMEE algoritmus természetesen más indexeléssel is működőképes. Más indexelés esetén úgy kell számolni a H és az L értékét, hogy a 3. lépésben a legnagyobb tartalmazott maszkot kapjuk. Az indexek számításához annak a függvénynek az inverzét kell használni, amivel kijelöltük az indexelés érték határait. Mivel a példában erre a hatvány függvényt használtuk, ezért az indexek értékét a kettesalapú logaritmussal kellett számolni, ami a hatvány függvény inverze.

Az IMEE algoritmus úgy is átírható, hogy a legkisebb tartalmazó maszkot használjuk mindig. Ehhez a 2. lépést úgy kell átírni, hogy L számításánál lefelé, H számításánál felfelé kell kerekíteni. Ilyenkor az algoritmus visszaadhat tiltott ételeket is. Ilyenkor valamelyik korábbi EHETEM-E algoritmussal tovább

szűrhetünk. Vagy az indexelést úgy kell beállítanom, hogy egy bizonyos hibahatáron belül adjon csak tiltott ételt vissza az algoritmus.

Az **indexelés hibája** az indexelt érték és a megszorításban lévő tényleges érték különbségének abszolult értékének és a tényleges érték hányadosa, amit százalékosan fejezünk ki. Példa:

> Legyen a megszorítás a következő: fehérje > 1,5dkg. Az IMEE algoritmus ehhez az L = 2 értéket rendeli. Ez a 2^(L-1) = 2 index határnak felel meg. Így az indexelés hibája: ABS(2 - 1,5) / 1,5 = 33,33%.

Az **indexelés hibájának maximuma** akkor adódik, amikor a tényleges érték a legkisebb, amire még az IMEE algoritmus ugyanazt az index értéket adja.

A most használt indexelés maximális hibája 100%, mert minden L értékre $\lim_{N->2^{(L-2)}}((2^{(L-1)}-N)/N) = 100\%$, ahol 2^(L-1) a legkisebb tartalmazó maszk által leírt megszorítás alsó határa, a 2^(L-2) pedig az előző indexelhető érték.

5.5. Fokozatosan finomított lista

Az IMEE algoritmus legérdekesebb alkalmazása, amikor fokozatos finomításra használjuk az algoritmust. Itt az algoritmusnak azt a tulajdonságát használjuk ki, hogy nem a pontos listát, hanem egy attól szűkebbet ad vissza, ugyanakkor ezt a szűk listát nagyon gyorsan.

Az internet világában gyakori feladat, hogy egy nagy információs halmaz közelítését nagyon gyorsan le lehessen tölteni. Ezt a következő algoritmus vázzal érhetjük el:

- Egy szűk halmazt gyorsan lekérdezni (LNTM).
- Ezt a szűk halmazt gyorsan eljuttatni a felhasználóhoz.
- Ha a felhasználó elidőzik a listán, akkor a lista finomítása:
    - LKTM – LNTM halmaz meghatározása.
    - Ezen halmazból leválogatni az eredeti feltételeknek megfelelő elemeket.
    - Leválogatás közben az új elemek folyamatos eljuttatása a felhasználóhoz.

Programozás technikailag ezt úgy érhetjük el, hogy a lista finomítását egy külön szál végzi a szerveren. Ezt a szálat csak akkor indítjuk, ha a felhasználó elidőz a listán. Ha a szál talál egy újabb elemet LKTM – LNTM halmazban, ami megfelel a megszorításoknak, akkor a szerver elküldi azt a felhasználó által használt klienshez.

Ez egyes elemek vizsgálatához felhasználhatom valamelyik EHETEM-E algoritmust.

6. Teszt eredmények

A tesztelés során az Oracle 11g adatbázis kezelő rendszert használtuk. Ezen belül tárolt eljárásokat, hiszen ezek gyorsabb adathozzáférést biztosítanak. Erre látunk példát ebben a cikkben [Rad2004].

A teszt adatbázist http://www.ars.usda.gov/Services/docs.htm?docid=20959 linkről töltöttük le. A következő tárolt eljárással készítettük el az indexeket.

```
create or replace
```

```
FUNCTION GET_INDEX( ORG_VALUE IN NUMBER, COLNUM IN NUMBER) RETURN NUMBER AS
vIND1 NUMBER; -- a tartományok változói
vIND2 NUMBER; vIND3 NUMBER; vIND4 NUMBER; vIND5 NUMBER;
vIND6 NUMBER; vIND7 NUMBER; vIND8 NUMBER; vIND9 NUMBER;
vIND10 NUMBER; vIND11 NUMBER; vIND12 NUMBER; vIND13 NUMBER;
vIND14 NUMBER; vIND15 NUMBER; vIND16 NUMBER; vIND17 NUMBER;
BEGIN
-- bemásoljuk az ABBREV_IND táblából az egyes tartományok értékeit
SELECT IND1, IND2, IND3, IND4, IND5, IND6, IND7, IND8, IND9, IND10, IND11, IND12, IND13, IND14, IND15, IND16, IND17 INTO vIND1, vIND2, vIND3, vIND4, vIND5, vIND6, vIND7, vIND8, vIND9, vIND10, vIND11, vIND12, vIND13, vIND14, vIND15, vIND16, vIND17
FROM ABBREV_IND WHERE COL_NUM = COLNUM;
-- if segítségével az index megállapítása
IF ORG_VALUE <= 0 THEN RETURN 262143; -- = 1111 1111 1111 1111 11
ELSIF ORG_VALUE <= vIND1 THEN RETURN 131071; -- = 0111 1111 1111 1111 11
ELSIF ORG_VALUE <= vIND2 THEN RETURN 65535; -- = 0011 1111 1111 1111 11
ELSIF ORG_VALUE <= vIND3 THEN RETURN 32767; -- = 0001 1111 1111 1111 11
ELSIF ORG_VALUE <= vIND4 THEN RETURN 16383; -- = 0000 1111 1111 1111 11
ELSIF ORG_VALUE <= vIND5 THEN RETURN 8191; -- = 0000 0111 1111 1111 11
ELSIF ORG_VALUE <= vIND6 THEN RETURN 4095; -- = 0000 0011 1111 1111 11
ELSIF ORG_VALUE <= vIND7 THEN RETURN 2047; -- = 0000 0001 1111 1111 11
ELSIF ORG_VALUE <= vIND8 THEN RETURN 1023; -- = 0000 0000 1111 1111 11
ELSIF ORG_VALUE <= vIND9 THEN RETURN 511; -- = 0000 0000 0111 1111 11
ELSIF ORG_VALUE <= vIND10 THEN RETURN 255; -- = 0000 0000 0011 1111 11
ELSIF ORG_VALUE <= vIND11 THEN RETURN 127; -- = 0000 0000 0001 1111 11
ELSIF ORG_VALUE <= vIND12 THEN RETURN 63; -- = 0000 0000 0000 1111 01
ELSIF ORG_VALUE <= vIND13 THEN RETURN 31; -- = 0000 0000 0000 0111 11
ELSIF ORG_VALUE <= vIND14 THEN RETURN 15; -- = 0000 0000 0000 0011 11
ELSIF ORG_VALUE <= vIND15 THEN RETURN 7; -- = 0000 0000 0000 0001 11
ELSIF ORG_VALUE <= vIND16 THEN RETURN 3; -- = 0000 0000 0000 0000 11
ELSIF ORG_VALUE <= vIND17 THEN RETURN 1; -- = 0000 0000 0000 0000 01
ELSE RETURN 0;
END IF;
END GET_INDEX;
```

A GET_INDEX egy beérkező érték (ORG_VALUE) és oszlopszám (COLNUM) alapján egy indexet ad vissza. Egy segéd táblát használ, az ABBREV_IND táblát. Ez tartalmazza az index határokat. Itt az 1. index úgy lett meghatározva, hogy minden összetevőnél megkerestük a legkisebb értéket. A következő index mindig az előző kétszerese. Mivel a legnagyobb és a legkisebb érték aránya egyik összetevőnél sem volt nagyobb 2^17-nél, ezért 17 index értéket használtunk.

```
create or replace
PROCEDURE INDEX_LOOP AS
   var ABBREV%ROWTYPE; -- ABBREV típusú változó
   CURSOR curso IS SELECT * FROM ABBREV ORDER BY NDB_NO; -- kurzor
BEGIN
   FOR var IN curso LOOP -- végigmegy a tábla összes során
    var.WATER_IND:= GET_INDEX(var.Water,1); -- index lekérése
    UPDATE ABBREV SET WATER_IND = var.WATER_IND WHERE NDB_NO =
var.NDB_NO; -- az aktuálisan vizsgált sor első eleméhez tartozó index frissítése
    var.Energ_Kcal_IND:= GET_INDEX(var.Energ_Kcal,2);
    UPDATE ABBREV SET Energ_Kcal_IND = var.Energ_Kcal_IND WHERE NDB_NO =
var.NDB_NO;
    …
   END LOOP;
END INDEX_LOOP;
```

Az INDEX_LOOP eljárás végigmegy az ABBREV tábla összes során, és frissíti a GET_INDEX függvény alapján az indexeiket. Itt csak az első két összetevő frissítését mutattuk meg. A többi összetevő frissítését töröltük, mert mindegyik nagyon hasonló, így könnyen megírhatóak a kihagyott sorok is. Ezek után az egyes x_IND oszlopokra egy bitmap indexet helyeztünk el.

Ettől függetlenül a WATER és minden egyéb összetevő oszlopra is elhelyeztünk indexet. Nyilván ezekre felesleges lett volna bitmap indexet tenni, mert nagyon sok különböző érték van bennük.

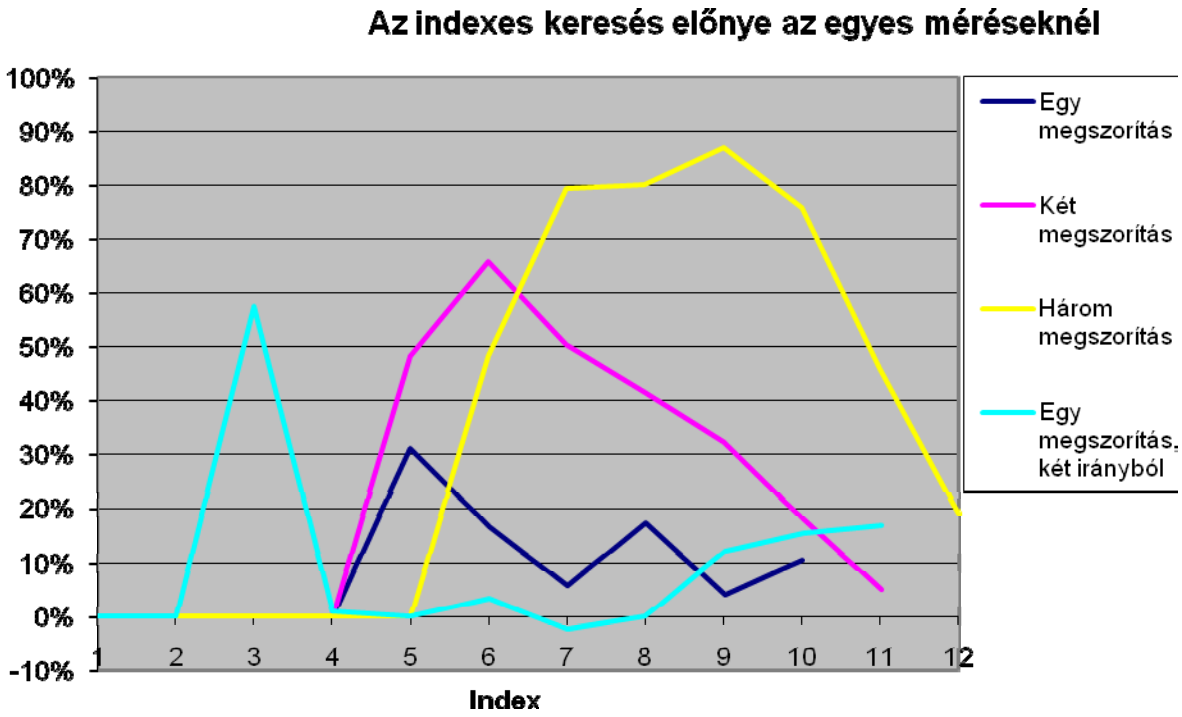
A tesztelt SELECT utasítások közül közöljük a legbonyolultabbat:

```
SELECT * FROM ABBREV WHERE WATER_IND >= POWER(2,9)-1 AND ENERG_KCAL_IND >=
POWER(2,9)-1 AND PROTEIN_IND >= POWER(2,9)-1;
```

```
SELECT * FROM ABBREV WHERE WATER <= (SELECT IND9 FROM ABBREV_IND WHERE
COL_NUM = 1) AND ENERG_KCAL <= (SELECT IND9 FROM ABBREV_IND WHERE COL_NUM =
2) AND PROTEIN <= (SELECT IND9 FROM ABBREV_IND WHERE COL_NUM = 3);
```

Az első SELECT bitmap indexet használ, a másik sima megszorítást. A második SELECT utasításban lévő belső SELECT visszaadja azt az értéket, ami az első SELECT utasításban használt feltételnek megfelel. Így a két SELECT teljesen ugyanazt adja vissza.

A tesztek során a következő eredményre jutottunk:



Az X tengelyen értéke a POWER(2,X)-1 képletbe lett behelyettesítve az első SELECT utasításban.

A 0% a második SELECT futási ideje. Ehhez viszonyítottuk az első SELECT futási idejét relatíve. A pozitív érték jelentése, hogy gyorsabb az első SELECT, a negatív tartomány jelentése, hogy lassabb. Az egyes összehasonlítások abban különböznek, hogy a SELECT utasítás WHERE záradékában hány megszorítás van.

Látható, hogy X = 9 és 3 megszorítás használatának esetében a bitmap indexet használó SELECT közel 10-szer gyorsabb, mint az eredeti megszorítást használó „sima" SELECT. Nagyon érdekes, hogy ez a különbség a bitmap index javára csak sok százezer rekordot tartalmazó adatbázis esetén volt megfigyelhető. Néhány ezer rekordból álló adatbázisnál a bitmap indexelés nem gyorsabb (de legalább nem is lassabb), mint a nem bitmap indexet használó lekérdezés.

Hogy nagyobb X értékre miért jobb a BITMAP index? Erre az lehet a magyarázat, hogy nagyobb X értékre több sort ad vissza a teszt során használt SELECT utasítás.

7. Összefoglalás

Ebben a cikkben nagyon alaposan átnéztük az EHETEM-E algoritmus legkülönbözőbb változatait. Ezek közül néhányat a Wit-Sys Zrt. munkatársai meg is valósítanak az eFilter pályázat keretében. Bizonyosan lehet még csiszolni ezeket az eljárásokat, miután a gyakorlatban megfigyeltük, milyen lekérdezések gyakoriak.

A BITMAP indexelés ilyen fajta használata más élelmiszerekkel foglalkozó alkalmazásokban is hasznosíthatók, főleg, ha mennyiségi megszorításokat kell figyelni. Ilyen alkalmazás lehet az árulistába történő keresés, ahol egy adott összegnél olcsóbb árukat keresünk. Ezt például egy ár összehasonlító oldal keretében kitűnően lehetne hasznosítani.

A cikkben ismertetett fokozatos finomítás egyelőre nincs implementálva. Erre valószínűleg csak egy következő projekt keretében kerül sor.

Irodalomjegyzék

[CI1998] C.-Y. Chan and Y. E. Ioannidis: Bitmap index design and evaluation, Proceedings of the SIGMOD ’98 International Conference on Management of Data, p. 355 – 366, 1998.

[EF2007a] Tibor Radványi, Gábor Kusper: Requirement analysis and a database model for the project EGERFOOD Food Safety Knowledge Center (invited talk), Proceedings of ICAI-2007, Volume I. 15-23, Eger, Hungary, January 2007.

[EF2007b] Kusper Gábor, Radványi Tibor: Az EGERFOOD élelmiszerbiztonsági tudásközpont projekt információs rendszerének kialakítása, Networkshop 2007 konferencia, Eger, 8 oldal, 2007.

[EF2007c] Kálmán Liptai, Gábor Kusper, Tibor Radványi: Cryptographycal protocols in the Egerfood Information System, Annales Mathematicae et Informaticae 34, ISSN 1787-5021, pp. 61-70, 2007.

[EF2008a] Radványi Tibor, Kusper Gábor, Kovács Emőd: Adatforgalom és mobilkommunikáció az Egerfood rendszerben, AgriaMédia 2008 konferencia, Kötet I. 100-107, 2008.

[EF2008b] Radványi Tibor, Kusper Gábor, Kovács Emőd: Kommunikáció az EGERFOOD élelmiszerbiztonsági projekt információs rendszerében, IF2008 konferencia, CD-kiadvány, ISBN 978-963-473-129-0, 10 oldal, 2008.

[EF2008c] Kusper Gábor, Radványi Tibor: Az EGERFOOD élelmiszerbiztonsági nyomkövető rendszer – Hogyan modellezzük a cégek munkafolyamatait, Networkshop 2008 konferencia, Dunaújváros, 8 oldal, 2008.

[In2005] W. H. Inmon: Building the Data Warehouse, Fourth Edition, Inmon-Wiley, ISBN: 978-0-7645-9944-6, 2005.

[LXLQ2009] Z. Lijuan, G. Xuebin, W. Linshuang, and S. Qian: Research on Materialized View Selection Algorithm in Data Warehouse, Proc. of IFCSTA09, p. 326 – 329, 2009.

[Rad2004] Tibor Radványi: Examination of the MSSQL server from the user's point view considering data insertion, Acta Academiae Pedagogicae Agriensis, p. 69-77, 2004.

[SM1985] I. Spiegler, R. Maayan: Storage and retrieval considerations of binary data bases, Information Processing and Management: an International Journal 21 (3), p. 233-254, 1985.

[WOS2006 K. Wu, E. J. Otoo, and A. Shoshani: Optimizing bitmap indices with efficient compression, ACM Transactions on Database Systems, Volume 31 Issue 1, p. 1-38. 2006.