EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR
MÉDIA ÉS OKTATÁSINFORMATIKAI TANSZÉK

A webstatisztika-készítés módszerei

diplomamunka

Juhász Péter Károly

Témavezetõ:
Illés Zoltán

Budapest, 2008.

Tartalomjegyzék

Előszó

A mai internet-központú világban nagy jelentőséggel bírnak a web-statisztikák, analízisek, azaz a látogatási adatok gyűjtése, feldolgozása, és ezek alapján különféle statisztikák előállítása, trendek megfigyelése, valamint a felhasználói viselkedésminták meghatározása és elemzése. Egyetlenegy komoly internetes üzlet vagy portál sem nélkülözheti ezen adatokat ahhoz, hogy maximalizálják a látogatószámukat valamint ezáltal a bevételüket. Az információnak nagyon nagy értéke van, de ugyanúgy fontos annak megbízhatósága, és rendelkezésre állási ideje is.

A világ egyre jobban az internet felé tolódik, mostanában már egy komolyabb cég sem nélkülözheti a vállalati web-lapot, ahol bemutathatja tevékenységi körét és termékeit. Egyre többen üzemeltetnek web-áruházat vagy valamilyen internettel összefüggő üzleti vállalkozást. Az egyre inkább növekvő tempójú világban az egyre nagyobb az érdeklődés az autómatikusan a felhasználó igeényeihez és persze az üzelemtető piaci céljaira idomuló weboldal iránt. Az ilyen dinamikus web-alkalmazások készítéséhez elengedhetetlen a naprakész, esetleg valós idejű statisztika illetve felhasználói modellezés.

A dolgozat célja, hogy körüljárja az előzőekben felvázolt okok miatt létfontosságú webstatisztikák készítéséhez szükséges matematikai és technikai hátteret, valamint bemutassa az ezekben használt algoritmusokat és módszereket. Továbbá cél egy kis történelmi áttekintés is a bemutatott technikákról, azok szerepéről a múltbeli és jelenlegi internetes világban, valamint a különféle technikák használatának, előnyeinek és hátrányainak a bemutatása is. Ezen utóbbiakat azért, mert befolyásolták a statisztikakészítés módszeretit.

A dolgozat mellé egy példa web-es alkalmazást is készítettem, ami gyakorlatban szemlélteti a probléma egy lehetséges megvalósítását. Bővebben bemutatom és elemzem az alkalmazásban előforduló érdekesebb részleteket, de természetesen kitérek az enyémtől eltérő egyéb megoldások előnyeinek és hátrányainak a bemutatására is.

A dolgozat további célja, hogy ezen alkalmazáson keresztül szemléltesse a fejlett web-es technológiákat, ide értve az AJAX megoldásokat, az adatbázisra épülő objektumközpontú programozási technikákat a szerveroldalon, a JavaScript segítségével a DOM adatmodellre épülő adatnyerési technikákat, valamint a háttérben meghúzódó algoritmusokat, amiket a program felhasználói viselkedéselemzésre használ.

1. fejezet

Technikai áttekintés

1. Bevezető

A web-fejlesztés világában különféle trendek figyelhetők meg időről időre. Ezek között előfordulnak zsákutcák, de persze az előrelépés a gyakoribb. Vannak trendek, amiket egy-egy technika megjelenése inspirál, vannak, amiket a divat és némelyeket egyéb külső tényezők. Ezek a trendek befolyásolják dolgozatom témáját képező web-statisztika készítés módszereit, ezért ebben a fejezetben körüljárom ezeket.

Egy dolgot leszögezhetünk. Azoknak, akiknek van valamilyen web-oldala, nagyon fontos tudniuk, hogy az mennyire népszerű. Az elején ennek amolyan "kisgyerekek között versengés" szerepe volt, ami az emberi természetből adódik, de mostanában, amikor az internetet elöntötte a reklám, anyagi vonzata is van. Vannak a kicsik: web-naplót[1] írók, zárt körű közösségi portál- illetve közösségi wiki[2] üzemeltetők. Vannak helyi érdekeltségű cégek, amelyek a termékeiket mutatják be illetve árusítják az interneten. Továbbá persze vannak a nagy világcégek, globális érdeklődésre számot tartó wiki-k és az internetes áruházak is, amelyeknél marketing szempontból nélkülözhetetlen az internetes statisztika. Egyre többen akarnak megélni a hálózatból, az üzleti szempontok jelentősen nőttek az utóbbi időben.

A fejezet további részeiben bemutatom az internetes tartalom fejlődését, kitérek a fontosabb technikai újításokra, úgy mint az AJAX, a CSS és a sablonok. Azért ezekre térek ki részletesebben, mert az AJAX-os megoldások nagy hatással voltak és vannak a web-statisztika készítésre, ugyanis új megoldások kitalálását tették szükségessé. Az ezirányú fejlődés még csak napjainkban indult és ezért nap mint nap új megoldások látnak világot.

[1] blog

[2] A wiki (illetve WikiWiki) a hypertext rendszerek egyik fajtája, vagy pedig maga a szoftver, ami ennek készítését lehetővé teszi. A wikiwikiweb olyan webhely, amely wiki rendszer szerint, ennek felhasználásával működik, vagyis lehetővé teszi azt, hogy a felhasználók (vagy általános esetben bárki) a laphoz új tartalmakat adjanak, vagy azon tartalmat módosítsanak. Más szóval a wiki egy olyan program, amelynek számos különböző implementációja létezik. Segítségével egész weboldalak is működtethetők (nem feltétlenül lexikon jelleggel), de alkalmazható a hagyományos fórumok helyett is a látogatók tapasztalatainak, véleményeinek strukturáltabb megjelenítésére. Kitalálója, Ward Cunningham szerint „a legegyszerűbb online adatbázis”; gyakran használják csoportos munkavégzés támogatására, közösségépítésre.

Ahogy egyre tolódik a hangsúly az online iroda felé, úgy egyre több és több dolog kerül a világhálóra és egyre interaktívabb lesz a tartalom. Ezek az dinamikus tartalom által felvonultatott újítások nagyban befolyásolják a statisztikagyártás módszereit. A CSS-ről és a sablonokról az általam bemutatott megoldás miatt beszélek részletesebben, mivel egy korszerű web-es alkalmazás elkészítéséhez mindkettőre szükség van és ezek is befolyásolták az internet fejlődését valamint azt, hogy miért és hogyan alakult ki az amit ma láthatunk.

2. Az Internetes tartalom fejlődése

Az Internet a korai szakaszban (1990-es évek eleje) csak különálló oldalak gyűjteménye volt. A kezdeti években közel 100%-os növekedést produkált évente, sőt '96 -'97 táján ennél is nagyobbat. Az információ egyszerűbb elérése érdekében először létrejöttek linkeket tartalmazó oldalak majd a keresőmotorok. Ez utóbbiak is jelentősen befolyásolták a mai web-oldalak képét. A web-oldalakat a fejlesztők elkezdték strukturálni. A kezdetben meglévő ide-oda hivatkozásos kavalkádból létrejöttek a menü vezérelt web-oldalak, később a *keretes*[3] oldalak. Itt egy keretben találhatók a web-oldal elemei, főként a menü fejléc, esetleg lábléc és a tartalom. Ennek az adta az apropóját, hogy az ilyen jellegű oldalak jobban áttekinthetőek és elég mindig csak a tartalmi részt letölteni, nem kell mindig az egész oldalt. Ez utóbbi a korai időkben nagyon fontos volt, hiszen akkortájt az emberek jelentős része még analóg telefonvonalon keresztül internetezett. Egy speciális fajtája a keretes meggondolásnak a *belső keret*[4] (1996), amikor egy kis doboz tartalmát lehetett a web-oldalon belül változtatni. Ez még a mai AJAX-os világban is használt elem, de egy mellékvágánynak tekinthető a fejlődésben.

Egy nagy átalakulást az internetes keresőmotoroknak köszönhetünk. A web-lap fejlesztők rájöttek, hogy a keretes megoldást nem preferálják a keresőrobotok (ezek azok az alkalmazások, amik járják az internetet és adatbázisba mentik a web-oldalak tartalmát, kapcsolatait, hogy aztán azt elérhetővé tegyék a keresőkben), ezért elkezdett a világ újból a teljesen letöltődő oldalak világa felé elmozdulni. Ennek az egyre nagyobb elérhető sávszélesség most már nem szabott határt. Ekkortájt jelentek meg az objektumközpontú programozást és a sablonos megvalósítást felvonultató oldalak, amik szintén ezt az irány erősítették.

Mivel azonban az adatnak még mindig át kell utaznia a hálózaton, megjelent az oldal villogásának a gondja, amit a keretes megvalósítás valamelyest elfedett. Ez azt jelenti, hogy amikor a felhasználó kattintott a web-oldalon, és megjelent az új web-oldal, ami esetleg csak egy kevés tartalmi részben különbözött, akkor a teljes oldal újratöltődött, amit még a web-böngészők esetleges lassúsága is szaggatottabbá tett. Erre egy megoldást az akkortájt már népszerűsödő kliens oldali programozás, ami konkrétabban a JavaScript, szolgáltatta. Egy népszerű technika volt az oldalak háttérben való letöltése és összerakása, ezzel ugyan megmaradt a „villogás" de legalább egyben jelentek meg az oldalak.

[3] frame
[4] iframe

A nagy, mai napokban is folyó átalakulást az AJAX[5]-os technikák megjelenése és egyre szélesebb körű elterjedése jelenti. Az AJAX interaktív web-oldalak létrehozására szolgáló technika. A web-lap kis mennyiségű adatot cserél a szerverrel a háttérben, így a lapot nem kell újratölteni minden egyes alkalommal, amikor a felhasználó módosít valamit. Ez növeli a honlap interaktivitását, sebességét és használhatóságát. Az AJAX elterjedését a szebb és rugalmasabb megjelenítést lehetővé tevő CSS[6] stílusleíró nyelv megjelenése segítette elő.

Persze mindig vannak visszatartó erők is. Esetünkben a különféle böngészők közötti különbségek a leginkább gátló hatásúak, főként az Internet Explorer okozott eddig nagy fejfájást az internetes fejlesztőknek és arculattervezőknek. Ezen különbségek főként a szabványok be nem tartásából adódnak. Vannak kezdeményezések ezen különbségek legyőzésére, ilyen például az Acid tesztsorozat, ami különféle aspektusokra fókuszált teszteket állít böngésző programírók elé.

Itt tartunk jelenleg. A tendenciák arra mutatnak, hogy egyre több dolgot visznek az emberek az internetre. Vannak próbálkozások internetes operációs rendszerek fejlesztése irányába is, amikor a felhasználó leül a gép elé, rámegy a web-oldalra, belép és mindazt elérheti ott, mint amit mostanában a gépén, de ezt bárhonnan. A dokumentumait, képeit, mindent. Továbbá egyre inkább szerepet kap az internet a csoportmunka elősegítésében. Itt is jelennek meg a jobbnál jobb megoldások. Továbbá persze az interaktivitás is nagy húzóerő, ami előreviszi a fejlődést. Arra mutat a fejlődés, hogy egy sokkal egységesebb, szabványosabb és csillogóbb internet felé haladunk.

A továbbiakban kicsit bővebben írok az egyes fontosabb dolgoktól, mint az AJAX, CSS és a sablonok.

3. Az AJAX

Az AJAX avagy *Asynchronous JavaScript and XML* egy jelenleg népszerű web-fejlesztési technika, ami több független komponenst ötvöz interaktív web-oldalak létrehozására. Nagyobb interaktivitást kínál, mint az eddigi megoldások, kevesebb adatforgalommal, mivel nem tölti újra a teljes web-oldalt, csak annak a szükséges részeit, ezzel gyorsítva a válaszidőt és növelve a felhasználói élményt.

Ezt úgy éri el, hogy háttérben kommunikál a szerverrel, így nem zavarja a felhasználót az oldal statikus részeinek nézésében. Míg eddig egy-egy táblázat frissítésénél az egész oldal újratöltődött, az AJAX technikának köszönhetően csak a változó részek frissülnek a háttérben és a felhasználó csak az adott rész frissülését veszi észre.

Az adatmozgatást a JavaScript nyelven megírt kliensoldali programok végzik általában, bár nem kötelező módon XML formátumban.

AZ AJAX platformfüggetlen technika, a mai összes olyan népszerű operációs rendszer és web-böngészővel használható, amelyek támogatják a JavaScript-et és a DOM[7] objektummodellt, ami egy nyelv- és böngészőfüggetlen HTML és XML leíró modell.

[5] Asynchronous JavaScript and XML
[6] Cascading Style Sheets
[7] Document Object Model

3.1. Az AJAX történelme

Az AJAX kifejezést 2005-ben James Garrett használta először. Az idáig vezető út azonban már majd egy évtizeddel azelőtt kezdődött. 1996-ban jelent meg az IFRAME az Internet Explorer 3-as verziójában, majd a LAYER elem a Netscape 4-es verziójában 1997-ben. Ez már lehetővé tette, hogy az oldal különálló részei különféle forrásból származzanak és, hogy ezen részek forrását JavaScript-tel megváltoztassuk. 1998-ban a Microsoft bevezette a Remote Scripting (MSRS) fogalmát, ami Java applet segítségével valósította meg az adatfrissítést és a klienssel JavaScript-tel kommunikált. Majd az Internet Explorer 5-ben bevezette az XMLHttpRequest objektumot, aminek a segítségével már JavaScript-ből tudta megoldani az adatlekérést. A közösségi fejlesztés először a microsoft.public.scripting.remote hírcsoporton keresztül zajlott, majd blog-okon, és 2002-re kifejlődött a tényleges Java applet-et leváltó technika, ami most már az összes népszerű böngészővel működött.

3.2. A használt technikák

Az AJAX a következő technikák ötvözete:

- (X)HTML és CSS a web-oldal tényleges megjelenítésért,
- a DOM modell, amin keresztül a JavaScript manipulálja a web-oldalt,
- ax XMLHttpRequest objektum, ami az aszinkron adatcserét valósítja meg a szerverrel,
- és az XML, amiben az adatot formázzák a szerver és a kliens között (ehelyett bármely más formátum alkalmazható).

Mint látszik a legfontosabb összetevő az XMLHttpRequest objektum, a többi kiegészítő csak a szebb és egyszerűbb megjelenítést segíti elő.

3.3. Előnyök

Az AJAX technológia sok esetben megkönnyíti a programozó munkáját, mivel lehetősége van, hogy sok hasonló web-lap helyett csak egyet készítsen el, majd azt különféle tartalommal töltse fel. Mindezt a felhasználó elől rejtve.

További előnyök közé lehet sorolni, hogy az AJAX-ra épülő web-oldalak gyorsabban töltődnek be, mivel az oldal keretét és az egészet vezérlő JavaScript file-t csak egyszer kell letölteni, valamint az oldalhoz szükséges további adatok közül is mindig csak azt a kis darabot kell letölteni, amire éppen kíváncsi a felhasználó.

A szerverek kihasználtságát is csökkenti, mivel kicsi adatmorzsákat tölt le csak és a megjelenítéshez szükséges esetleges számításokat már a kliens tudja végezni.

Továbbá ez a technika a programozót tisztább, könnyebben szeparálható kódok írására ösztönzi. Ez alatt a következő komponensekre osztást értem:

- a formázatlan adat, általában XML formátumban,
- a web-oldal HTML kerete
- a megjelenítési stíluslapok CSS-ben
- és végül a web-oldal funkcionális része: szerver oldali programok, kliens oldali programok és az ezek közti kommunikáció.

3.4. Hátrányok

Egyik legnagyobb hátránya, hogy JavaScript -képes böngésző szükséges hozzá. Mivel vannak olyan böngészők, amik erre nem képesek illetve a felhasználó akár le is tilthatja azt, ezzel részlegesen vagy teljesen működésképtelenné téve az oldalt.

Továbbá biztonsági rés is keletkezhet a JavaScript letiltása által, mivel így kiiktatásra kerül az AJAX hívások általi adatellenőrzés, ezért erre a programozónak fel kell készülnie.

A fehasználói élményt károsíthatja, hogy a *Vissza* gomb és a *Könyvjelzők* AJAX-os oldalokon nem a megszokott módon működnek. Ennek a megoldására a leggyakoribb megoldás láthatatlan IFRAME-ek alkalmazása azért, hogy a böngésző mégis tudjon valamit eltárolni a *Vissza* gomb részére. Valamint a *Könyvjelző* használhatóságára az URL `#` utáni részét szokták használni. Ezek a megoldások azonban nem tökéletesek, például az IFRAME nem része az XHTML 1.1 szabványnak, a W3C[8] as `object` elemet ajánlja helyette.

A háttérben lévő adatforgalom és annak az esetleges lassúsága meglepheti a fehasználót, mert közben az oldal nem csinál semmit (az AJAX nélküli világban ilyenkor jelezte a böngésző, hogy adat letöltés folyik és az oldal újraépítését is látta a fehasználó). A programozónak erre fel kell készülni és jelezni a felhasználó felé, hogy történik valami a háttérben.

A kereső motorok indexelési technikáját is keresztülhúzza az AJAX, mivel azok nem hajtják végre a JavaScript-et, ezért oldaltérképet kell nekik készíteni.

A különféle böngészők eltérő viselkedése is nehézzé teszi az AJAX-os alkalmazások elkészítését, mivel nemcsak a JavaScript értelmezésében, de a DOM modell implementálásában is lehetnek különbségek, amik érthetetlen hibákat képesek produkálni. Ezért nagy figyelmet kell fordítani a tesztelésre.

Végül a diplomamunkám témáját képező web-statisztika készítést is részben keresztülhúzza az AJAX, sőt a hagyományos web-szerver naplókon alapuló kimutatásokat is összezavarhatja, mivel ilyenkor általában csak az oldal egyes részeit töltik újra, de van, ahol majdhogynem az egészet is. Ezt persze a web-es statisztika generálásánál lehet enyhíteni azzal, ha az AJAX hívás által generált oldalba is beillesztik a számláló kódot.

4. CSS

A CSS, azaz a Cascading Style Sheets egy, a dokumentum megjelenítéséért felelős leíró nyelv. Főképpen web-oldalak megjelenítésénél használják, de bármilyen XML dokumen-

[8] World Wide Web Consortium

tumnál használható. Mobil alkalmazások és elektronikus könyvek megjelenítésénél is alkalmazzák.

Azért találták ki, hogy a dokumentum tartalmát (HTML) és a megjelenítését (CSS) elkülönítsék egymástól. Ez az elkülönítés nemcsak a fejlesztőknek ad könnyítést (gondolok itt arra, hogy akár a grafikusi és a programozói munkakört is el lehet így különíteni), hanem különféle eszközökön is lehetőséget ad a különféle megjelenítésre (PDA, PC, nyomtató, telefon). Sőt a képernyőolvasóknak is lehet külön stílust definiálni a gyengén látók segítésére.

A specifikációt a W3C kezeli, és a text/css MIME típus[9] 1998 márciusától van használatban, amit a 2318-as RFC[10] vezetett be.

Az előzőleg létező nyelvektől eltérően (DSSSL, FOSI) a CSS lehetőséget ad, hogy egy dokumentum több stílusfile által egyidejűleg legyen kontrollálva.

4.1. A CSS történelme

A CSS irányába való törekvés már az 1970-es években megkezdődött az SGML[11]-el.

Ahogy a HTML fejlődött, fejlődött vele a megjelenítés is, hogy a készítők minél változatosabb oldalakat tudjanak csinálni. Ez nagyobb szabadságot adott az arculattervezőknek, de a fejlesztők életét megnehezítette, mivel az egyre bonyolódó oldalakat egyre nehezebb volt átlátni és karbantartani. Főleg a böngészők közti különbségek keserítették meg a fejlesztők életét, amikor megpróbálták mindenhol megvalósítani az arculattervezők elgondolásait.

A helyzet enyhítésére 9 stílusleíró nyelv közül választott ki kettőt a W3C, amikből a CSS létrejött. Az elsőt Hőkon Wium Lie fejlesztette ki 1994 októberében és Cascading HTML Style Sheets (CHSS) névre hallgatott. A másikat Bert Bos fejlesztette az akkor készűlő Argo névre hallgató böngészőjéhez, és Stream-based Style Sheet Proposal (SSP) volt a neve. Lie és Bos együtt fejlesztette ki a mai CSS-t.

Hőkon Wium Lie a Chicagoi "Mosaic and the Web" konferencián mutatta be az elképzelését 1994-ben, majd Bert Bos 1995-ben. Abban az időben alapították a W3C-t, ami rögtön felfigyelt a CSS-re és felkarolta azt. A projekt vezetője továbbra is Hőkon és Bert volt, de csatlakozott hozzájuk Thomas Reardon a Microsoft-tól és még sokan mások. 1996 decemberében publikálták a CSS első verzióját, ezzel hivatalos lett.

A HTML, CSS és DOM fejlesztését a HTML Editorial Review Board (ERB) csoport végezte. 1997-ben az ERB szétvált három kisebb csoportra: HTML Working group Dan Connolly (W3C) vezetésével, DOM Working group Lauren Wood (SoftQuad) vezetésével és a CSS Working group Chris Lilley (W3C) vezetésével.

A CSS Working Group 1997 november 4-én publikálta a CSS második verzióját. A harmadikon 1998-ban kezdtek el dolgozni és a fejlesztése még a mai napig is tart.

[9] A Multipurpose Internet Mail Extensions, rövidítése MIME, az e-mail formátumot kiterjesztő internetszabvány. A MIME szabvány által definiált tartalomtípusok az e-maileken túlmenően is nagy fontossággal bírnak, például a HTTP hasonló kommunikációs protokollok használatakor.

[10] Request for Comments

[11] Standard Generalized Markup Language

Bár a CSS1 1996 óta létező szabvány, a fogadtatása nem volt gördülékeny. Az Internet Explorer 3-at ugyanabban az évben adta ki a Microsoft, ami még korlátozottan, de támogatta a CSS-t. További 3 év telt el, amire valaki majdnem teljesen támogatta azt. Az Internet Explorer 5.0 Macintosh verziója 2000 márciusában jelent meg és 99%-ban támogatta a CSS1-et, ezzel lekörözve az Opera-t, ami már majdnem másfél éve a CSS-t legjobban támogató böngésző volt. A többi böngésző ezek után gyorsan követte őket, és implementálták a CSS1-et és ezzel együtt a CSS2-t is részben, de 2006 júliusáig egyetlen böngésző sem támogatta a CSS2-t teljesen.

Emiatt az összevissza támogatottság miatt a CSS-nek nagyon nehéz dolga volt az elfogadás terén. Még akkor is, amikor már majdnem mindenki támogatta a CSS-t, mivel azok hibákkal voltak tűzdelve és nem volt konzisztens a megjelenítés.

A W3C ezen problémák miatt felülvizsgálta a CSS2-t és kiadta a CSS2.1-et, ami kicsit közelebb helyezkedett el a jelenlegi megvalósítási állapothoz. Az olyan részeket, amiket egyetlen böngésző sem implementált, kidobták és néhol a specifikációt a megvalósítás irányába mozgatták. A CSS2.1 2004 február 25.-én debütált, de 2005 június 13.-án visszahívták és csak 2007 július 19.-én lett újra ajánlott.

2006-ban még voltak olyan web-szerverek, amelyek a `css` kiterjesztősű file-okat tévesen `application/x-pointplus` MIME típussal szolgálták ki. Mivel a Net-Scene, PointPlus Maker néven forgalmazta a programját, ami PowerPoint file-okat Compact Slide Show file-okká konvertált (`css` kiterjesztéssel). Mivel a Netscape Navigator 3.0 rendelkezett ennek a megjelenítéséhez szükséges beépülő modullal, ezért ez nem jelenítette meg a külső CSS file-okat.

4.2. Variációk

A CSS egymásra épülő szintekből és profilokból áll. Minden szint az előzőre épül új kiegészítések hozzáadásával, ezek a CSS1, CSS2 és a CSS3. A profilok a szintek részhalmazai, ilyenek a különféle hardware-ekre és a mobil gépekre szánt verziók. Ugyancsak ide tartozik a printereknek szánt profil is.

A CSS1 a következő dolgokat tartalmazza:

- font tulajdonságok;
- szöveg, háttér illetve egyéb elem színei;
- szöveg, kép illetve táblázat elhelyezési módok;
- szöveg manipulációk (térközök, sorok);
- margók illetve belső térközök;
- ezen tulajdonságok csoportosíthatósága.

A CSS2 a következő újításokat tartalmazza:

- abszolút, relatív illetve fix pozicionálás;
- új font tulajdonságok;
- jobbról balra író nyelvek támogatása.

A CSS2.1 a nem támogatott dolgokat hagyta el és egy-két újat vett hozzá, amit a böngészők már támogattak.

A CSS3 még jelenleg is fejlesztés alatt áll, várhatólag jobban modularizált lesz és az XHTML szabvánnyal jobban össze fog fonódni.

4.3. Hátrányok

Azon túl, hogy a CSS megreformálta a web-fejlesztést, van néhány hátránya is.

Nem konzisztens viselkedés A különféle böngészők eltérő módon jelenítik meg a CSS-t. Ez leginkább implementációs hibáknak köszönhető, illetve hiányzó támogatásnak. Ezen különféle, leginkább JavaScript-es, illetve CSS-es ügyeskedéssel lehet valamennyire úrrá lenni, de még így is szinte képtelenség pixel pontosan ugyanazt az eredményt hozni minden böngészőben.

Szülő választás képtelensége A CSS nem kínál megoldást egy bizonyos tulajdonságokkal rendelkező elem kiválasztására, ennek erőforrástakarékosság és megjelenítési okai vannak.

Függőleges elhelyezési korlátok Az elemek vízszintes elhelyezése könnyű, ellenben a függőleges elhelyezésre a CSS nem biztosít elég eszközkészletet, ezért az nehézkes, sőt néha lehetetlen a kívánt eredmény elérése.

Műveletek hiánya Nem lehet az egyes paraméterek értékét műveletekkel megadni (például: 15% - 5em - 2px)

Elnevezési kavarodások Van, hogy több tulajdonság állítása is ugyanazt az eredményt adja. Valamint vannak nem logikus elnevezések is (táblázat mezőjének nincs `margin-*` tulajdonsága, csak `border-spacing`).

Nem lehet több háttere egy elemnek A CSS3-ban erre lesz lehetőség.

Az elemek formája nem választható Jelenleg csak dobozokból lehet építkezni, nem lehet lekerekített sarkakat csinálni. (Illetve csak ügyeskedéssel, azaz több doboz egymáshoz illesztésével érhető el a kívánt hatás.)

Változók hiánya Nem lehet változókat deklarálni és azokat használni a stílusok megadásánál.

Többhasábos szedés hiánya Jelenleg a 2 vagy több hasábos szedést elemek úsztatásával lehet megoldani, ami néha nem az elvárt eredményt szolgáltatja.

5. Sablonok

A sablonok használata a web-programozásban fontos szerepet játszik, mivel sablonok segítségével könnyebben, gyorsabban és főleg áttekinthetőbben lehet internetes alkalmazásokat létrehozni.

A sablonok használata több elemből épül fel, ezek a

Sablon motor egy olyan program elem, ami a sablonból az adatok segítségével felépíti és megjeleníti a web-oldalt;

Adatok ezek lehetnek adatbázisból, file-ból illetve szerveroldali program által előállítottak, ezek töltik meg tartalommal a sablont;

Sablon ez adja a keretet az egész oldalhoz, a benne lévő változóknak a sablon motor ad értéket.

Egy tipikus sablon motor a következő szolgáltatásokat adja a programozó kezébe: változók és függvények deklarálásának lehetősége, szövegműveletek, külső file-ok beszúrásának lehetősége és ciklusok valamint elágazások használhatósága.

5.1. Előnyök

A sablonok egyik fő előnye a web-oldalak tömeggyártása közben jelentkezik. Képzeljünk el egy céget, aminek a web-oldalali statikusak és miután a tervek alapján a programozó legyártotta őket, megváltozik a cég címe, ami minden oldal alján szerepel. Sablonok használata nélkül a programozónak minden oldalon egyesével kellene azt kijavítania, sablonok segítségével elég csak egy helyen javítania. Persze ez igaz a stílusra is nem csak a szövegre, főleg, ha CSS-el is dolgozik a fejlesztő.

Már a fenti példa is rámutat arra, hogy a sablonokkal a fejlesztő jobban el tudja különíteni az oldal részeit, és így ha valamelyik részt módosítani kell, akkor azt a többitől függetlenül teheti meg. Ezek a részek általában a fejléc, lábléc, menü és a tartalom.

A fejlesztési gyakorlatot is hatékonyabbá lehet tenni a sablonok használatával. Régen általában egy fejlesztő készítette az oldalt, aki egyben volt programozó és stílustervező is. Sablonok használatával ezek a szerepek szétválaszthatóak, külön ember készítheti a sablont, annak segítségével külön ember a stílust, és külön ember illetve emberek a programot mögötte.

A programozó akár be is zárhatja a forrást, mivel az arculattervezőnek arra nincsen szüksége a munkájához. Így akár külsősökkel is végeztethetik az esetleg üzleti titoknak számító programon alapuló web-oldal megjelenítését.

5.2. A sablonrendszerek fajtái

A piacon rengeteg sablonrendszer van, zárt- illetve nyílt forrású és ezeken belül is jelentősen eltérő megvalósításúak.

Több dolog szerint lehet kategorizálni őket. Nyelvi megvalósítás szerint lehetnek: beágyazott, illetve eseményvezérelt; egyszerű illetve programozható; illetve már az előbb említett forráskód szerinti nyílt illetve zárt.

Lehetnek *szerveren kívüli* sablonrendszerek, ezek nem mások, mint web-szerkesztő programok, amik támogatják a statikus web-oldalak sablonos felépítését. Ilyenek például az: Nvu, BlueFish, FrontPage és a Dreamweaver. Több program tudja a web-oldalakat egyenesen a szerverre menteni, például FTP segítségével.

A másik faja a *szerver oldali* sablonrendszer. Itt speciális program illetve programkomponensek dinamikusan állítják elő a web-oldalt.

6. A web-statisztika

A web-statisztika készítés célja a látogatói adatok gyűjtése, rendszerezése, majd azokból következtetések levonása, mostanában már a felhasználói viselkedéselemzés is ide tartozik. Ennek a természetes emberi kíváncsiságon és hiúságon túl fontos szerepe van az üzleti életben is, ahol az üzleti célkitűzések megvalósítására tett erőfeszítések visszajelzéseként lehet azt tekinteni. Sok fajta statisztikai kimutatás készíthető kezdve az igazán egyszerűktől, ahol csak a napi találati számokat tüntetjük fel, a grafikonokon keresztül a komplex felhasználói modellek elkészítéséig. Úgy mint minden más is ez a témakör is hatalmas utat járt be az utóbbi időben. Az elején még csak egyszerű web-napló feldolgozó programok voltak, most meg már hatalmas űzleti vállakozások vannak, amelyek méregdrága programokat készítenek, vagy komplex online szolgáltatásokat kínálnak.

Persze nemcsak szigorúan a web-oldalak kattintásai tartoznak bele a web-statisztika készítés forrásai közé, hanem az e-mail forgalom és az esetleges tényleges eladási mutatók is szorosan összefonódnak a témával. De ez már inkább mutat a üzleti elemzők világa felé mint a programozókéhoz.

Mint az előbbi megállapításokból is látszik, ez a témakör nagyon szerteágazó, ezért nem is lehet célom, hogy töviről hegyire ismertessem azt, inkább csak a programozói szemszögből vizsgálom meg a témát. Koncentrálva az adatforrásokra, bemutatva a legalapvetőbb adattípusokat és egy kicsit jobban kitérve arra, hogy hogyan is lehet az adatokból kinyerni a leggyakoribb felhasználói útvonalakat, ezzel téve egy kicsi betekintést a felhasználói viselkedéselemzés világába.

6.1. Az adatgyűjtés

A web-statisztika gyártáshoz többféle adatforrást is lehet használni. Ebben a részben bemutatom ezen adatforrásokat, írok az előnyeikről és hátrányaikról.

Web-szerver naplók

A web-szerverek mindig is tárolták a tranzakciók naplóit. Elég hamar rájöttek az emberek, hogy ezt programmal feldolgozva tudják mérni az oldalak látogatottságát. Az 1990-es évek

elején még csak a találatokat nézték, mivel akkortájt egy oldal általában csak egy HTML file volt. Ahogy azonban egyre szaporodtak a web-oldalak (megjelentek a több file-ból álló web-oldalak) és egyre több kép jelent meg rajtuk, így ez a szám kezdte elveszteni a jelentőségét.

Az első igazi web-szerver napló analizátort a I/PRO adta ki 1994-ben.

Ekkortájt két mérőszámot vezettek be, ami az előzőknél precízebben jellemezte az oldalak látogatottságát. Ez az oldalmegnézés (annyival jobb, mint a találat, hogy csak a tényleges oldalmegjelenéseket tartalmazza, a befoglalt képek és egyebek nem számítanak) és a látogatás. Az utóbbi az egy felhasználó által egybefüggő találatok számát jelenti. Egybefüggő alatt azt értik, hogy két találatot nem választ el egy meghatározottnál nagyobb idő, ez általában 30 perc körül van. Ezen adatok még most is nagy jelentőséggel bírnak, de már más mérőszámok is nagy szerephez jutottak azóta.

Az 1990-es évek vége felé, ahogy a keresőmotorok egyre népszerűbbek lettek, úgy váltak a web-szerver naplók egyre *zajosabbá*. A proxy szerverek és a dinamikus IP címek megjelenése is tovább nehezítette a felhasználói azonosítást. Minderre a web-statisztika gyártók válasza a cookie[12]-k bevezetése volt.

A gépek memóriájának és háttértárolóinak a növekedése magával hozta az adatok cache-elését, azaz eltárolták egy bizonyos ideig a látogatott web-oldalakat és amikor a felhasználó (illetve akárki más, ha proxy-ról van szó) ugyanazt az oldalt kéri, akkor azt nem kérik le újra a szerverről, ezzel takarékoskodva a sávszélességgel. Ezt a cache-elést szerver oldalról lehet tiltani, de ez nem ajánlott, mert az terheli a szervert és csökkenti a felhasználói élményt.

Előnyei A legfőbb előnyei ennek az eljárásnak a következők:

- A web-szerverek maguktól is készítenek naplókat, nem kell semmi extra dolgot telepíteni hozzá. Csak egy olyan program kell, ami ezt feldolgozza, és ebből nagyon sok van. A legelterjedtebb formátumok a Common Log Format és az Extended Log Format.

- A naplók megbízhatóak, nem lehet kompromittálni őket például a JavaScript letiltással vagy a képek megjelenítésének letiltásával.

- A naplók a web-oldalt üzemeltető szerveren vannak, ezért utólag egyszerűen lehet más statisztikagyártó programra váltani, nem függenek egy külső szolgáltatótól.

- A keresőmotorok pókjai[13], azaz az olyan programok, amik feltérképezik az internetet, hogy aztán elérhetőek legyen a keresőben, is nyomot hagynak a naplókban, holott

[12] A cookie egy kis szöveges üzenet a a böngészőnek, amit az eltárol a benne meghatározott ideig, majd a web-oldal, amely eltároltatta a cookie-t, az ki tudja olvasni a benne foglalt adatokat. Leggyakrabban jelszóval és felhasználónevek tárolására használják, amikor a felhasználó bepipálja bejelentkezéskor, hogy *"emlékezz rám"*. Mint látható lesz a web-statisztikákban is fontos szerepet játszik, a felhasználók követésében.

[13] Search engine spider

azok nem töltik le általában a képeket és nem futtatnak JavaScript-et. Így ez az adat használható kereső optimalizálásra.

- A naplók a sikertelen oldallekérésekről is tartalmaznak adatokat.
- Régi adatokat is könnyű felhasználni, gondolok itt a statisztikagyártó program melletti döntés előtti adatokra, mivel a szerverek vannak, hogy több évig is megőrzik a naplókat.
- Nem kell tűzfalak miatt aggódni.
- Sávszélességet, befejezett és be nem fejezett letöltéseket is lehet figyelni.
- Mobil készülékről érkező látogatókat is számlálja természetesen.

Hátrányok

- A legnagyobb nehézség az egyes felhasználói un. session-ök azonosítása, azaz azon lekérdezések időben rendezett sorozata, amik egy meghatározott felhasználótól érkeznek, mivel a web-szerverek nem tudnak köztük különbséget tenni. A lehetséges megoldásokról majd később lesz szó.
- Ha cookie-t akarunk elhelyezni a felhasználó gépén, akkor módosítani kell a weboldalt.
- Nem lehet annyi adatot kinyerni a böngészőből, mint az oldalmegjelöléssel.
- Meg kell szűrni az adatokat a felhasználás előtt, eltávolítva abból a képekre és egyéb, nem tényleges oldallekérésekre utaló hivatkozásokat.
- Ha az oldal cache-ben vagy progy-ban megvan, akkor annak az újrakérését észre sem vesszük.
- Nem lehet Flash-es illetve JavasSript-es eseményeket rögzíteni.
- A statisztikagyártó program telepítését és frissítését is saját magunknak kell végezni.
- A naplókat saját magunknak kell tárolni és archiválni.

Oldalmegjelölés

Az előzőekben leírt hátrányok (proxy, cache) egy új technológia kifejlődéséhez vezettek. Ez az oldalmegjelölés (Page tagging) vagy más néven Web bogarak (Web bugs) használata.

Az 1990-es évek elején sok web-oldalon tűntek fel számlálók, amik kis képek formájában a látogatásszámot mutatták. Az 1990-es évek végére ezek láthatatlan képekké fejlődtek, amik JavaScript által jelentek meg és a kép lekérése által plusz információkat juttattak a számlálást végző szerverhez.

Megjelentek a cookie-k is, amik a felhasználókat azonosították a látogatás időtartalmára, valamint a visszatérő látogatókat is ezek követték.

Az AJAX-os technológiák terjedésével a nulla méretű képek kihagyhatóvá váltak, mivel azok nélkül az `XmlHttpRequest` objektum használatával is lehetővé vált a szerverhez kérést intézni. Bár ezen megoldás sok buktatót tartogat az előzőekben már bemutatott böngészők közti eltérések miatt.

Előnyei Fő előnyei ennek a megoldásnak az alant felsoroltak:

- A JavaScript mindig lefut, amikor az oldal betöltődik, így nincs baj a cache-elés miatt, sem a proxy szerverekkel.
- JavaScript segítségével több adatot lehet gyűjteni, mint például a hivatkozó oldal címét, a böngésző képességeit, képernyőfelbontást és még sok mást.
- Könnyebb cookie-kat elhelyezni a felhasználó gépén.
- Akkor is alkalmazható, ha nem a saját web-szerveren van a web-oldal, hanem csak a tárhelyet béreljük.
- Még Flash események is követhetők vele.
- Nem kell a web-oldal üzemeltetőjének a statisztika gyártó programmal foglalkoznia (frissítéseket telepíteni).

Hátrányok

- Hibás vagy kihagyott kód beillesztés az oldalba adatvesztéshez vezet.
- Tűzfalak esetleg megakadályozhatják a JavaScript-es visszanyúlást.
- Nem lehet sávszélességet mérni illetve a sikeres letöltéseket számolni, mivel a szerverhez való visszahivatkozás akkor jön létre, amikor az oldal letöltődik.
- Keresőmotor pókokat nem lehet monitorozni.

Hibrid megoldások

Vannak termékek, amik mind a két megoldást ötvözik és így pontosabb statisztikát tudnak szolgáltatni, mintha csak az egyiket vagy másikat alkalmaznánk. Az első ilyen megoldás 1998-ban jelent meg.

Egyéb megoldások

Léteznek még más, de kevésbé elterjedt megoldások is. Vannak, ahol a statisztikagyártó programot a web-szerverbe integrálják, de használnak még adatfolyam monitorozókat[14] is, hogy a teljes adatfolyamot felvegyék és azt elemezzék. Ennek a módszernek egyedisége a stop gomb megnyomásának az észlelése. Továbbá használnak még módosított klienseket is, bár ez a felhasználó közreműködését követeli meg. Vannak, akik speciális proxy szervereket állítanak be, hogy az gyűjtse az adatokat. Ezek akár módosítják is az elküldött adatokat, ezáltal megjelölve a hivatkozásokat és így követve a felhasználót.

6.2. Figyelt adatok

Itt azon adatokat illetve mérőszámokat sorolom fel, amiket a statisztikagyártó programok szoktak szolgáltatni. Néhol írok az adott adat kiszámításáról illetve megszerzésének módjáról is.

Találat A web-szerver által kapott találatok száma. Beletartozik minden egyes file, amit a web-szerver kiszolgál a klienseknek, nem csak a web-oldalak. Mivel egyes web-oldalak nagyon sok kis különálló file-ból állnak, és esetleg AJAX-os hívásokkal frissítik az adatot, ezért ez a szám nagyon félrevezető lehet. Így inkább a web-oldal komplexitását és a tényleges látogatottságát egyszerre adja vissza. Ezt a számot csak a web-szerver naplókból lehet megállapítani.

Oldalszám A tényleges oldalletöltések számát adja meg. Ez már jól tükrözi az oldal népszerűségét. Mind oldalmegjelöléssel, mind web-szerver naplókkal elő lehet állítani ezt a számot.

Látogatószám A látogatók számát próbálja meg megadni. Egy látogatás a beazonosított felhasználó időben egymáshoz közeli találatait tartalmazza. A közeli itt általában 30 percet jelent. Ha ennél több idő telik el két találat között, akkor az már új látogatásnak számít az adott felhasználótól.

Új / visszatérő látogató Ha egy látogató már előzőleg látogatta a web-oldalt, akkor visszatérőnek tituláljuk, különben új látogató. Az előző pontban leírt 30 es intervallum kimaradás után is már visszatérő az adott látogató, ha előtte új is volt.

Egyedi látogatók A látogató számot adja vissza, de egy látogatónak csak egy látogatását számolja.

Visszatérési idő A látogatók utolsó és utolsó előtti látogatása között eltelt idő. Általában napokban mérik.[15]

Látogatáshossz Az átlagos értéke a látogatások közbeni oldalletöltések számának.

[14] Packet sniffer
[15] visitor recenity

Ugrási arány[16] Az olyan látogatások százalékos arányát adja vissza, amik csak egy oldalmegnézésből álltak. Fontos adat, mert arra lehet következtetni belőle, hogy a látogatók hány százaléka a csak véletlenül az oldalra keveredett illetve a tényleges látogató. Ez nagyban függ azonban az oldal típusától is. A technikai blog-ok és egyéb olyan oldalak, ahol a felhasználókat segítő kisebb cikkek vannak, sok találatot kapnak keresőktől. Az ilyen látogatók általában csak arra az adott dologra kíváncsiak, amit megtaláltak, és nem navigálnak tovább az oldalon. A tesztjeim szerint egy átlagos közösségi oldal 25% körül kap véletlen találatokat. Egy Wiki típusú oldal, amit nagyon beindexeltek a keresők, de mégis csak egy kis réteget érdekel, az 50% körül kap, és a technikai blog-ok 80-90% körül.

Óránkénti átlagos látogató A megadott időszakra az egyes órákban, mint intervallumokban mért átlagos látogatószám. Szokták még napi, heti és havi bontásban is megadni. Megfigyelhető belőle, hogy a látogatók honnan nézik az oldalt. Ha napközben nézik, akkor azt inkább munkahelyként lehet azonosítani, míg ha az esti órákban, akkor inkább otthonról.

Utolsó látogatók Általában minden program képes arra, hogy a legutolsó n darab látogató adatait részletesen megmutassa. Persze az adatbázisból illetve a naplókból az összes kinyerhető, de erre nem szoktak lehetőséget kínálni.

Látogatók gépére jellemzők eloszlása Értve ide a böngésző, operációs rendszer, képernyőfelbontás, cookie-k tárolásának illetve JavaScript illetve Java apple tárolásának futtatásának a képessége. Ezen adatokat általában százalékosan adják vissza, változó, hogy milyen pontossággal (egyszerűen csak Windows illetve Windows XP, 2000, ...).

Aloldalak A népszerű aloldalak nevei illetve azok látogatottságának százalékos eloszlását adják meg.

Keresőszavak illetve keresőkifejezések A keresőoldalakról érkező látogatók által használt keresőszavak (ilyenkor a keresőkifejezéseket szavanként bontják) illetve a keresőkifejezéseket adja meg. Általában a leggyakoribb n darabot sorolják fel az onnan érkező találatok számával.

Hivatkozók Azon oldalak címei, ahonnan a látogató átkattintott, általában elkülönítik a keresőket illetve az egyéb oldalakat. Itt is mint a keresőszavaknál a leggyakoribbakat szokás megjeleníteni, de sok program lehetőséget ad az összes lekérésére is.[17]

Forrás ország Általában képesek rá a programok, hogy az egyes látogatókról beazonosítsák, hogy mely országból érkeztek. Ezt az IP cím alapján teszik, mert mivel azok szigorúan ki vannak osztva és akár még a várost, sőt a nagyobb cégeket illetve egyetemeket

[17] Az általam írt program az itt gyűjtött adatokból intelligensebb módon azt is ki tudja mutatni, hogy hova került fel az utóbbi időben az oldal, illetve, hogy mely hivatkozások lettek inaktívak vagy újra aktívak.

is be lehet vele azonosítani[18]. Az interneten fellelhetők mind fizetős, mind ingyenes adatbázisok, amik az IP-t és az országot rendelik egymáshoz.

Oldaltípusok A látogatások alapján a programok képesek besorolni az oldalakat aszerint, hogy melyeken kezdődnek illetve érnek véget a látogatások, de vannak, ahol ezeket csak százalékosan adják meg, mint az aloldalak népszerűségét.

Látogatás időbeli hossza A látogató első és utolsó oldallekérése alapján ezt is többékevésbé ki lehet számolni és ezt is szokás megjeleníteni. Mivel a látogatások közti 30 perces rés csak amolyan heurisztikusan választott adat, így ha a látogatás közben meglátogatott hivatkozások közti eltelt átlagidőket szorozzuk fel a látogatás hosszával, pontosabb értéket kapunk.

Robotok adatai A web-szerver napló alapú programok képesek a web-robotokról is adatokat adni, úgymint azok eredete (szolgáltatója) százalékos eloszlása, az általuk adott találatok aránya, a bejárt oldalak száma és nevei.

Könyvjelzők Egyes programok képesek a felhasználók általi könyvjelzőket is azonosítani és arról is információt adni, hogy hányan tettek az oldalra könyvjelzőt.

Látogatói útvonal Némely programok képesek analizálni a felhasználók viselkedését[19] és azt szöveges és/vagy grafikus formában (általában gráffal) reprezentálni.

7. A felhasználói viselkedéselemzés

Ebben a részben bemutatom az angol irodalomban *sequential pattern mining* problémaként ismert - a továbbiakban szekvenciális minta bányászat problémakör - formális hátterét, röviden a történelmét, valamint a mérföldkőnek számító algoritmusok működését.

A szekvenciális minták kinyerése a web-statisztikák létrehozásán túl fontos lehet a következő feladatok megoldásánál is, hogy adatokat nyerjenek ki nagy adatbázisokból: protein illetve DNS szekvenciák keresése, vásárlói analízis, tünetek és betegségek ok-okozati viszonyának felderítése és még sok egyéb.

7.1. A probléma

Adott események egy véges halmaza $E = \{e_1, e_2, \ldots, e_n\}$, ezek lesznek a hivatkozott weboldalak. Egy $s = \langle e_1, e_2, \ldots, e_m \rangle$, ahol $\forall i \in [1, m] : e_i \subset E$, s egy sorozat, ez írható $s = (e_1)\,(e_2)\ldots(e_m)$ alakban is, amennyiben (e_i) egy elemű, a zárójel elhagyható. Egy esemény többször is előfordulhat egy sorozatban. Egy sorozat hossza a benne lévő eseményhalmazok száma, ha egy sorozat l hosszú, akkor l-sorozatnak is hívjuk. A sorozatok nekünk az egyes felhasználók által bejárt útvonalak lesznek. Egy $S_a = a_1 a_2 \ldots a_n$ sorozat részsorozata egy

[18] Például az összes 157.181.*.* alakú cím az ELTE tulajdona.

[19] Ennek miként jéről bővebben lesz szó még a következő fejezetben.

$S_b = b_1 b_2 \dots b_m$ sorozatnak, ha létezik $1 \leq i_1 < i_2 < \dots < i_n \leq m$, hogy $a_1 = b_{i_1}, a_2 = b_{i_2}, \dots, a_n = b_{i_n}$. Ezt $S_a \sqsubseteq S_B$-vel jelöljük.

Az input sorozat adatbázis, ISA $\{sid, S\}$ párokból áll, ahol *sid* a sorozat azonosító és S a sorozat. Az adatbázis méretét $|ISA|$ jelöli. Azt mondjuk, hogy az $\{sid, S\}$ tartalmaz egy S_a sorozatot, ha S_a részsorozata S-nek, azaz $S_a \sqsubseteq S$. Egy S_a sorozat egy ISA adatbázisbeli abszolút támogatottsága alatt azt a számot értjük, amennyi ISA beli $\{sid, S\}$ párosra igaz, hogy S_a részsorozata S-nek, ezt $sup_{ISA}(S_a)$-val jelöljük. A relatív támogatottság alatt $sup_{ISA}(S_a) / |ISA|$-et értünk.

Megadott támogatottsági szint (*minsup*) mellett egy S_a sorozat gyakori sorozat ISA-ban, ha $sup_{ISA}(S_a) \geq minsup$. Ha S_a gyakori sorozat és nem létezik olyan S_b sorozat, aminek részsorozata és S_b támogatottsága megegyezik S_a támogatottságával, azaz $\nexists S_b, S_a \sqsubset S_b, sup_{ISA}(S_a) = sup_{ISA}(S_b)$, akkor S_a-t gyakori zárt sorozatnak nevezzük.

Vannak algoritmusok, ahol *minsup*-nak százalékos értéket adnak meg. Itt akkor azt jelenti, hogy az ISA beli sorozatok hány százalékában kell minimum előfordulni a mintának.

sid	S
1	CAABC
2	ABCB
3	CABC
4	ABBCA

1.1. táblázat: Példa

1. Példa
A 1.1 táblázatban látható input sorozat adatbázisban 3 különböző elem található: A, B és C. Az adatbázis mérete $|ISA| = 4$. Legyen $minsup = 2$. Ekkor a következők a gyakori sorozatok: A:4, AA:2, AB:4, ABB:2, ABC:4, AC:4, B:4, BB:2, BC:4, C:4, CA:3, CAB:2, CABC:2, CAC:2, CB:3, CBC:2, CC:2, (a kettőspont utáni számok a támogatottságot jelentik). Illetve a következő gyakori sorozatok a zártak: AA:2, ABB:2, ABC:4, CA:3, CABC:2, CB:3. Például a CBC:2-t a CABC:2 szorította ki a zárt sorozatok közül, mivel $CBC \sqsubset CABC, sup_{ISA}(CBC) = sup_{ISA}(CABC)$.

A problémának többféle megközelítése is van. Az első, amikor az összes gyakori sorozatra vagyunk kíváncsiak, aminek a támogatottsága nagyobb, mint az előre meghatározott támogatottsági szint. A második, amikor csak a zárt sorozatokra vagyunk kíváncsiak ezek közül. A harmadik, amikor nincs meghatározott támogatottsági szint, hanem a k darab leginkább támogatott sorozatot keressük.

7.2. Történelem

A szekvenciális minta bányászat először 1995-ben bukkant fel Agrawal és Srikant *Mining sequential patterns*[3]. A problémát a következőképpen vezették be: *adott sorozatok egy halmaza, ahol minden sorozat elem-halmazokból áll, és adott egy felhasználó által meghatáro-*

zott minimális támogatási szint. A szekvenciális minta bányászat feladata, hogy megtalálja az összes olyan részsorozatot, amelyeknek az előfordulási száma nagyobb, mint a minimális támogatási szint.

A probléma megoldásának kétféle csoportja ismert, az első az asszociációs szabályokat alkalmazó algoritmusok, a második a fa struktúrákat használja illetve Markov láncokat.

Az alábbiakban a következő algoritmusokat ismertetem részletesebben: Apriori (1994)[5], GSP (1996)[7], FreeSpan (2000)[6], BIDE (2004)[8]

Az ismertetett algoritmusoknál igyekeztem megtartani az eredeti dokumentumokban használt példákat és jelöléseket, hogy az esetleges azokkal történő összevetés minél inkább egyszerűbb legyen.

7.3. Az Apriori algoritmus[5]

Az Apriori algoritmus [5] az asszociációs szabályokra épül.

Egy $X \Rightarrow Y$ hozzárendelést asszociációs szabálynak hívunk, ha $X \in ISA$, $Y \in ISA$ és $X \cap Y = \varnothing$. Minden szabályhoz egy bizonyossági mérték tartozik és azt fejezi ki, hogy *ha egy sorozat tartalmazza X-es akkor az c%-ban tartalmazza Y-t is.* (Nálunk ez azt jelenti, hogy ha a felhasználó, aki meglátogatta az X web-oldalt akkor c%-ban meglátogatta az Y web-oldalt is.

Az ilyen asszociációs szabályok megtalálására már voltak algoritmusok, mint például az AIS és az SETM, de az Apriori volt az első, amit direkt web-naplók analizálására használtak.

Az asszociációs szabályok megtalálásának feladata két részre bontható:

- Meg kell találni az összes gyakori eseményhalmazt.
- Az előzőleg megtalált eseményhalmazok támogatottsága alapján meg kell konstruálni az asszociációs szabályokat. Például, ha AB és $ABCD$ két gyakori eseményhalmaz, akkor az $AB \Rightarrow ABCD$ szabály bizonyosságát az

$$\frac{\text{támogatottság}\,(ABCD)}{\text{támogatottság}\,(AB)}$$

  formulával számolhatjuk ki. Ha az eredmény nagyobb, mint *minsup*, akkor a szabályt megtartjuk.

Az Apriori algoritmus az első problémára ad megoldást.

Az algoritmushoz feltesszük, hogy a sorozatokon belüli események lexikografikusan rendezve vannak. (Mint látszik ebben az algoritmusban nem használjuk ki a hivatkozássorozatok időbeli rendezettségének voltát, de a későbbi algoritmusok között lesznek olyanok, amik ezt használják. Ebből látható, hogy az Apriori algoritmus és vele együtt az asszociációs szabályok nem erre lettek kitalálva, csak egy már meglévő algoritmust idomítottak a problémához.)

Az algoritmus első lépésnek megállapítja a gyakori 1-sorozatokat (ez itt ugyan csak rendezett halmazt jelent, de azért, hogy összhangban legyen az elnevezés a 7.1. részben leírtakkal, itt is ezt használom). Ezt a következő módon csinálja:

- Két k-sorozat egyesítése, ahol ha p és q a két sorozat és $p_1 = q_1, \ldots p_{k-2} = q_{k-2}$ és $p_{k-1} < q_{k-1}$, akkor az új sorozat $p_1, \ldots, p_{k-2}, p_{k-1}, q_{k-1}$ lesz.
- Majd töröljük az olyanokat, amiknek van olyan $k-1$ hosszú részsorozata, ami nem $(k-1)$-sorozat.

2. Példa
Legyenek a $(k-1)$-sorozatok a következők: 123, 124, 134, 135 és 234. Az egyesítés után az 1234 és a 1345 sorozatok keletkeznek. Ezek közül az 1345-öt töröljük, mivel az 145 nincs a $(k-1)$-sorozatok között, így csak az 1234 marad.

Ezek után a végrehajtás ciklikusan folytatódik, a k-adik ütemben az algoritmus két dolgot csinál. Először is a $(k-1)$-edik ütemben generált $(k-1)$-sorozatok segítségével legenerálja a k-sorozat *jelölteket*. Majd a második részben átnézi az adatbázist, hogy megállapítsa ezen jelöltek támogatottságát, akiknél a támogatottság nagyobb, mint a *minsup* ,az bekerül a k-sorozatok közé. Ha már nem tud többet generálni az algoritmus leáll.

Az algoritmus második lépésében lévő törléshez hasítótáblát alkalmaz, hogy gyorsítsa a tagsági vizsgálatot nagy sorozatszám esetén.

A C_k jelölthalmazokat hasítófában vannak ábrázolva úgy, hogy ha a fa leveleiben a sorozatok vannak, akkor a közbenső csúcsokban hasítótáblák. Új sorozat beillesztésénél addig megyünk lefele a fában, amíg a levelekhez nem érünk, miközben a belső csúcsokban a hasítófüggvény segítségével döntjük el, hogy merre menjünk tovább, alkalmazva azt a sorozat k-adik elemére, ha a k-adik szinten vagyunk a fában. Ha a fában egy levelében az elemszám meghalad egy előre definiált küszöbértéket, akkor azt belső csúccsá alakítjuk. Ezen fa segítségével már könnyen le lehet generálni egy t sorozat összes részsorozatát. Úgy, hogy ha levélben vagyunk ellenőrizzük, hogy mely elemek vannak benne t-ben, és azokat hozzávesszük az eredményhalmazhoz. Ha belső csúcsban vagyunk, ahova úgy jutottunk, hogy i-vel hasítottunk, akkor rekurízve minden olyan elemmel hasítunk, ami t-ben i-t követi. A gyökérben t minden elemével hasítunk. Ez így azt csinálja, amit akarunk, mivel mindig csak azokat az elemeket hagyja ki, amik nincsenek t-ben.

Az algorimus nagy hátránya, hogy rengeteg memóriát fogyaszt, mivel a jelölteket számon kell tartania és a már megtalált gyakori mintákat is tárolnia kell.

7.4. A GSP algoritmus[7]

A GPS (Generalized Sequential Patterns) algoritmust[7] ugyanaz a Srikant és Agrawal alkotta 1997-ben, akik az előző Apriori algoritmust két évvel korábban. Mind az Apriori algoritmus és annak variációi, mind a GSP algoritmus is a generál és tesztel módszert valósítják meg. Ennek több hátránya is van:

- Nagyszámú jelöltet generál a minták keresése során. Például, ha két 1 hosszú gyakori sorozatunk van, a és b, akkor ezekből 4 2 hosszú jelöltet generálnak: aa, ab, ba és bb. Ha 100 van, akkor 10000 darab 2 hosszú és 1000000 3 hosszú jelöltet generál és ellenőriz le.

- Többször olvassa végig az adatbázist. Ha l hosszú a leghosszabb szekvenciális minta, akkor az összes megtalálásához $l+1$-szer kell az adatbázist végigolvasni (az $l+1$-edik menetben nem találunk újat, így ott állunk le.

Az algoritmus az alapfeltevéseken felül felteszi, hogy létezik egy $\mathcal{T}$ erdő[20], ami „az egy" hozzárendelésekkel osztályozza az elemeket. Bevezeti továbbá az *ős* fogalmát, azaz egy $\hat{x}$ az x elem őse, ha létezik út $\hat{x}$-tól x ig $\mathcal{T}$ tranzitív lezártjában.

Továbbá azt is feltesszük, hogy minden elemhez egy időbélyeg is tartozik. Az algoritmus használ még egy előre definiált minimális-rést és maximális-rést, valamint egy csúszóablak méretet.

A csúszóablak méret a támogatottsági szint számításánál gyengíti a definíciót annyival, hogy egy $d = \langle d_1 \dots d_m \rangle$ input sorozat növeli az $s = \langle s_1 \dots s_n \rangle$ sorozat támagatottságát, akkor és csak akkor, ha $\exists l_1 \leq u_1 < l_2 \leq u_2 < \dots < l_n \leq\leq e_n$, hogy $s_i \in \cup_{k=l_i}^{u_i} d_k$ és tranzakcióidő (d_{u_i}) − tranzakcióidő $(d_{l_i}) \leq$ ablakméret, ahol $1 \leq i \leq n$.

A minimális- és maximális-rés szigorítja ezt a definíciót a következő két megkötéssel: tranzakcióidő (d_{l_i}) − tranzakcióidő $\left(d_{u_{i-1}}\right) >$ minimális-rés, ahol $2 \leq i \leq n$ és tranzakcióidő (d_{u_i}) − tranzakcióidő $\left(d_{l_{i-1}}\right) \leq$ minimális-rés, ahol $2 \leq i \leq n$. Azaz az input sorozat és a prefix sorozat elemeinek az időbeli eltérésére ad alsó és felső korlátot.

Az algoritmus többször megy végig a bemenetére adott input sorozatokon. Első menetben meghatározza a gyakori elemeket. Ezek alapján máris meghatározta a gyakori 1-sorozatokat. A további menetekben az algoritmus az input sorozatokon kívül az előző menetben meghatározott gyakori sorozatokat is megkapja bemenetként, majd ezek segítségével legenerál azoknál eggyel hosszabb jelölteket és ezekre is kiszámolja a támogatottságot, hogy eldöntse, hogy gyakoriak-e. Ha egy lépésben nem tudott új jelölteket generálni, akkor leáll, mert megtalálta az összes gyakori szekvenciális mintát.

A jelöltgenerálás

L_k jelölje a gyakori k-sorozatok halmazát és C_k a jelölt k-sorozatok halmazát.

1. Definíció
Legyen adott $S = s_1 s_2 \dots s_n$ sorozat és C, ahol $C \sqsubseteq S$, C folytonos részsorozata *S-nek, ha a következő állítások közül bármelyik fenn áll:*

1. *C S-ből s_1 vagy s_n-ből egy elem elhagyásával keletkezik*
2. *C S-ből úgy jön létre, hogy S egyik legalább 2 elemű elemhalmazából elhagyunk egy elemet*
3. *ha C a C' folytonos részsorozata és C' az S folytonos részsorozata*

Ez alapján a jelöltgenerálás két lépésre osztható:

[20] Aciklikus gráf

Összekapcsolás Összekapcsolunk két elemet s_1-et és s_2-t L_{k-1}-ből, amennyiben s_1 első eleme elhagyásával kapott részsorozata megegyezik s_2 annak utolsó elemének elhagyásával kapott részsorozatával. Az új elem az s_1 és az s_2 utolsó elemének az összefűzésével kapott sorozat lesz.

Elhagyás Elhagyjuk az olyan sorozatokat, amiknek létezik olyan folytonos részsorozata, aminek a támogatottsága kisebb, mint *minsup*, azaz van olyan folytonos részsorozata, ami nincs benne L_{k-1}-ben.

Például $L_3 = \{\langle(1,2)\,(3)\rangle, \langle(1,2)\,(4)\rangle, \langle(1)\,(3,4)\rangle, \langle(1,3)\,(5)\rangle, \langle(2)\,(3,4)\rangle, \langle(2)\,(3)\,(5)\rangle\}$, a kapott sorozatok az ősszekapcsolás után: $;\{\langle(1,2)\,(3,4)\rangle, \langle(1,2)\,(3)\,(5)\rangle\}$, ezek közül a másodikat elhagyjuk, mivel az $\langle(1)\,(3)\,(5)\rangle$ folytonos részsorozata nincs L_3-ban.

Jelöltszámlálás

A menetek végrehajtása alatt sorban veszi az input sorozatokat és az összes benne megtalálható jelölt sorozatnak megnöveli 1-el a támogatottságát. Ehhez szükség van arra, hogy egy C jelölt sorozat halmazhoz és egy d inputsorozathoz meg tudjuk találni az összes olyan C-beli sorozatot, ami benne van d-ben.

A felhasznált adatstruktúra Hasító-fa struktúrát használ a tárolás optimalizálására. A fában egy csúcs vagy sorozatokat tartalmaz, ha levél, vagy egy, a következő szintre mutató hasító táblázatot, ha belső csúcs.

Ehhez a fához a következő módon lehet új jelölteket adni. Elindulunk a gyökértől, az n-edik szinten a jelölt sorozat n-edik elemére alkalmazzuk a hasító függvényt, amivel a $n+1$-edik szintre jutunk. Ha elérjük a levelek szintjét, beillesztjük oda a jelöltet. Amennyiben egy levél elér egy meghatározott nagyságot, akkor belső csúccsá alakítjuk.

Annak eldöntésére, hogy egy d sorozat tartalmaz-e egy jelölt sorozatot a következőt kell tenni. Elindulunk a fában a gyökértől, ott a hasító függvényt minden elemre alkalmazzuk d-ben. Majd rekurzíve az összes elemre a megfelelő halmazban. Mivel bármely s sorozat első elemének, amit d tartalmaz, benne kell lennie d-ben. Az összes d-beli elemmel hasítva csak azon sorozatokat veszítjük el, amik olyan elemmel kezdődnek, ami nincs benne d-ben Ha az adott csúcs belső csúcs, de nem a gyökér, akkor tegyük fel, hogy úgy értük el a csúcsot, hogy egy t tranzakcióidejű x elmere alkalmaztuk a hasítófüggvényt. Alkalmazzuk a hasítást a d összes olyan elemére, aminek a tranzakciós ideje benne van a t − ablakméret, $t + max\,(\text{ablakméret}, \text{maximum-rés})$ intervallumban. Majd folytassuk ezt rekurzívan. Amennyiben levél csúcsban vagyunk, ellenőrizzük az ott található összes s sorozatra, hogy d tartalmazza-e s-et.

Annak ellenőrzése, hogy az input sorozat tartalmaz-e egy sorozatot A feladat annak eldöntésére, hogy egy d input sorozat tartalmaz-e egy $s = \langle s_1 \ldots s_n\rangle$ sorozatot.

Az algoritmusnak két részre bontható, először s elemeit keressük sorban d-ben, amíg az éppen megtalált elem és az előző elem közötti eltelt idő kisebb, mint a maximális-rés.

Ha ennél nagyobb lenne a különbség, akkor az algoritmus átvált a második, visszafelé haladó fázisra. Ha nem találunk meg egy elemet, akkor d nem tartalmazza s-t. A második fázisban az algoritmus *visszaszedi* az egyes elemeket. Ha s_i a jelenlegi elem és az ideje t, akkor az algoritmus megkeresi az első olyan elemet, aminek az ideje $t - maximlis - rs$ utáni. Ez a felszedegetés rekurzíve visszagyűrűzhet, ha a felszedés után s_{i-2} és s_{i-1} közti rés nagyobb lesz, mint a maximális-rés. Ha minden rés rendben van, vagy ha elfogy a teljes sorozat, akkor az algoritmus visszavált az előrefelé való fázisba és keresi a következő s beli elemet d-ben. Ha egy elemet nem lehet visszaszedni, mert nincs amire lecseréljük, akkor az algoritmus leáll, mert d nem tartalmazza s-t.

Tekintsük például a 1.2. példában szereplő adatbázist. Legyen a maximális-rés 30, a minimális-rés 5, az ablakméret 0 és keressük az $1,234$ sorozatot. Az algoritmus először a 10-es tranzakcióidejű $1,2$ elemet találja meg, majd a 45-ös idejű 3-at, de mivel ezek közti rész $35 > 30$ felszedi $1,2$-t. A következő $1,2$-t $45 - 30 = 15$-ös idő után kezdi el keresni és 50-nél találja meg. Mivel csak egy elemünk van, nem kell a réseket ellenőrizni rekurzívan, így átlép az előrefelé fázisba. A 3-as elemet $50 + 5 = 55$-ős időtől kezdi el keresni és 65-nél találja meg, mivel a rés nem nagyobb, mint 30, így folytatja az előrefelé fázist, és keresi a 4-et, amit 90-nél talál meg, a rés itt is megfelelő, és mivel nincs több elem, az algoritmus sikeresen leáll.

tranzakció idő	elem
10	1,2
25	4,6
45	3
50	12
65	3
90	2,4
95	6

1.2. táblázat: Példa

A fenti algoritmusban lévő egy elem keresés gyorsítására a következő megoldást használják. Az ISA-ban szereplő azonos elemek (nem elemhalmazok) tranzakció idejét láncba fűzik, és ezt a láncot használják egy megadott intervallumba eső tranzakcióidejű elem megtalálására. A megfelelő t idő utáni elemhalmaz keresésénél az algoritmus sorban vizsgálja a láncokat, ha talál olyat, ahol a két idő különbsége kisebb vagy egyenlő, mint az ablakméret, akkor készen vagyunk, ha nem, akkor t-t az ablakmérettel megnöveli és újra próbálja a keresést.

Tekintsük megint a 1.2. példát, és tegyük fel, hogy az ablakméret 7 és, hogy a $2,6$ elemet keressük $t = 20$ után. Először megtalálja az algoritmus a 2-t 50-nél és a 6-ot 25-nél, de mivel ezek közti idő ≥ 7 égy t-t $50 - 7 = 43$-ra állítja és újra próbálkozik. A 6-ot újra megtalálja 90-nél, míg a 2 marad 50-nél. Még mindig túl nagy a hézag, így tovább növeli t-t most 88-ra. Az új menetben 2-t 90-nél, 6-ot 95-nél találja meg azokat, amik az ablakméreten belül vannak, így sikeresen leáll.

Az algoritmus tovább finomítható még a taxonómiák bevonásával, de mivel annak az én esetemben nincs jelentősége, így annak bemutatásától eltekintek.

Az előbb bemutatott GPS algoritmus nagyságrendekkel gyorsabb, mint az Apriori algoritmus. Ennek az oka a kevesebb jelölt ellenőrzése.

7.5. A WAP-mine algoritmus[1]

A WAP-mine algoritmus merőben új adatstruktúrát alkalmaz, amivel abban az időbe átmenetileg átvette a vezetést a leggyorsabb szekvenciális mintakereső algoritmusok között, ez az adatstruktúra a WAP-fa.

Az algoritmus a feladat egy speciális osztályára működik, ahol minden az inputsorozatokat alkotó elemhalmaz 1 elemű.

Az algoritmus alapja a következő megállapítás: ha e egy gyakori elem a P-re vonatkozó prefixek között az ISA-ban, akkor eP egy szekvenciális minta ISA-ban.

A WAP-fa

A fa feladata, hogy az ISA kétszeri átolvasása után (ez a kevés átolvasás a legnagyobb előnye az algoritmusnak, az összes többivel szemben) eltárolja az összes információt, amire az algoritmusnak a későbbiekben szüksége lesz.

A fa csúcsai elem: darabszám formátumúak, ahol az $elem \in E$ és a darabszám azt adja meg, hogy mennyi azzal az elemmel végződő olyan prefixű sorozat van ISA-ban, ahol a prefixet a gyökértől az adott elemig lévő út határozza meg.

A fa építése a következő módon folyik. Első lépésként ISA egyszeri átolvasásával kiszűrjük a nem gyakori elemeket. Ezek után a megmaradt, már szűrt sorozatokat beillesztjük a fába. A gyökérbe egy virtuális elemet teszünk üres elemmel és 0 darabszámmal. Egy új sorozat beillesztése a gyökérnél kezdődik, ha ott van olyan gyerek, ahol az elem megegyezik a sorozat első elemével, akkor megnöveljük annak a darabszámát és továbblépünk az adott csúcsra. Ott a sorozat második elemét hasonlítjuk össze a gyerekekkel, és így tovább a sorozat végéig. Ha olyan csúcsba érünk, ahol nincs megfelelő gyerek, akkor beillesztjük azt 1 darabszámmal.

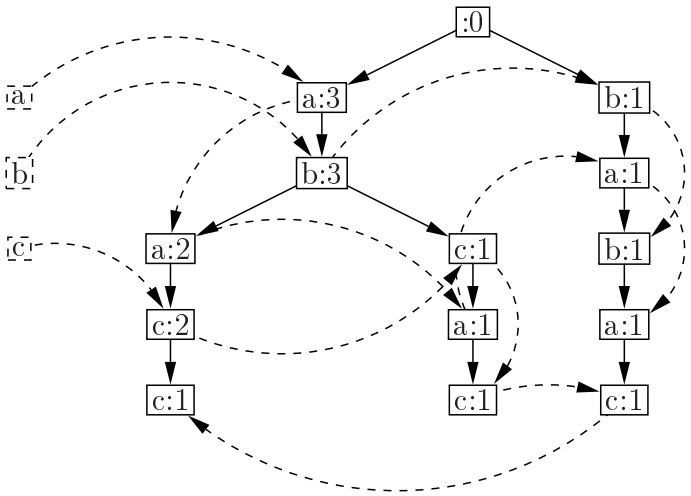
A fenti algoritmus szerint felépített fát kiegészítjük láncolásokkal, ami szükséges lesz majd az algoritmushoz. Minden, a fában lévő ugyanolyan E-beli elemet összeláncolunk. Továbbá azokat e_i-láncnak hívjuk, ahol $e_i \in E$. Például: a-lánc. Ezeknek a láncoknak a fejelemét egy külön táblába gyűjtjük össze, amit *fejtáblának* nevezünk.

Egy példa adatbázisból (1.3. példa) felépített WAP-fa látszik a 1.1. ábrán.

Az így felépített WAP-fa tárol minden, a bányászáshoz szükséges adatot. A fa maximális magassága a leghoszabb inputsorozat hossza, a maximális szélessége az ISA-ban lévő elemek száma.

sid	S
1	abdac
2	eaebcac
3	babfaec
4	afbacfc

1.3. táblázat: Példa



1.1. ábra: A 1.3. példában lévő adatbázis alapján generált fa. A szaggatott élek az e_i-láncokat reprezentálják.

A WAP-mine algoritmus

1. Tétel
Bármely gyakori e_i elemre az összes olyan gyakori szekvenciális mintát meg lehet látogatni, ami tartalmazza e_i-t, ha bejárjuk az e_i-láncot.

2. Definíció
A WAP-fában a gyökértől egy e_i csúcsig (e_i már nem tartozik bele) tartó sorozatot prefix sorozatnak *nevezünk. Az e_i csúcs darabszámát* a prefix sorozat darabszámának *hívjuk. Ha egy G és egy H is prefix sorozata e_i-nek (lehetséges, mivel e_i többször is előfordulhat egy láncon), és G részsorozata H-nak, akkor G-t a H* sub prefix sorozatának, *H-t a G* szuper prefix sorozatának *hívjuk. Egy e_i elem prefix sorozatának a* saját darabszáma[21] *a prefix sorozat darabszáma mínusz az összes szuper prefix sorozatának a saját darabszáma. Például: $a:3 \leftarrow b:3 \leftarrow a:2 \leftarrow b:2 \leftarrow c:1$ sorozatban a baloldali b saját darabszáma $3-2$, mivel 2 a 3-ból a jobboldali b miatt került bele a címkébe.*

2. Tétel
Egy e_i végű G sorozat elemszáma megegyezik az e_i összes G szuper prefix sorozatának a

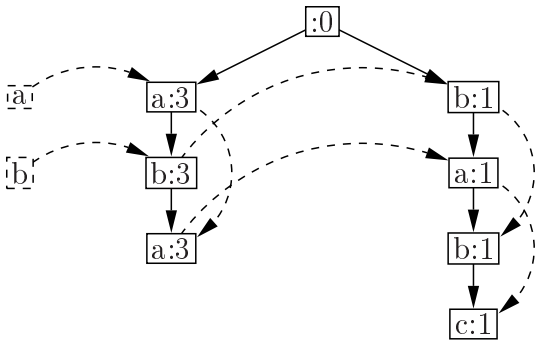
[21] unsubsumed count

saját darabszámával.

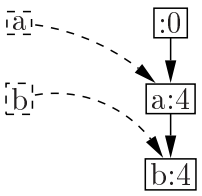
Ezen definíció és tételek alapján már meg lehet konstruálni a *feltételes keresés* algoritmusát, ami előállítja a szekvenciális mintákat. A feltételes keresés a prefixeket használja a keresési tér szűkítésére, így minél hosszabb már az adott prefix, annál jobban gyorsul az algoritmus.

A tényleges algoritmus a következő. Ha a fa csak egy ágból áll, akkor adjuk vissza a csúcsok összes sorrendtartó kombinációját. Ha több ágból áll, akkor visszaadandó halmazt inicializáljuk üresre, majd tegyük bele az összes fa -beli elemet. Ezek után az összes e_i fa -beli elemre (az e_i láncok segítségével) hajtsuk végre a következőket. Konstruáljuk meg az e_i*-feltételes szekvenciális mintákat*, azaz a $\mathcal{PS}|e_i := \{$az e_i és csak az e_i prefix sorozatai$\}$. Ha ez a halmaz nem üres, akkor építsük fel a e_i *feltételes WAP-fáját* az $\mathcal{PS}|e_i$-beli sorozatokból. Majd hajtsuk végre az algoritmust rekurzíve erre a fára. Majd az ezáltal visszaadott sorozatok elejéhez fűzzük hozzá e_i-t és tegyük bele a visszaadandó halmazba. Ahogy a rekurziókkal végzünk, megkapjuk az összes szekvenciális mintát.

Tekintsük az 1.1. ábrán lévő WAP-fát. Tegyük fel, hogy *minsup* 75%. A feltételes keresést kezdjük a c elemmel. A c-feltételes szekvenciális mintái: $aba : 2, ab : 1, abca : 1, ab : -1, baba : 1, abac : 1, aba : -1$. Az 75%-os *minsup* miatt csak az a és a b elemek a gyakoriak, így a 1.2. ábrán lévő feltételes WAP-fát kapjuk.



1.2. ábra: Az e_i feltételes WAP-fája



1.3. ábra: Az e_i feltételes WAP-fája

Rekurzíve az $\mathcal{PS}|ac = ab : 3, b : 1, ac : 1, b - 1$. Először ac feltételes WAP-fáját építjük, ez az 1.3. ábrán látható. Mivel ennek csak 1 ága van, így a szekvenciális minták, amiket meghatároz: $ac, aac, bac, abac$ és ac.

Az algoritmus hasonlóan folytatódik a többi ágon is.

7.6. A FreeSpan algoritmus[6]

A FreeSpan (Frequent pattern-projected Sequential pattern mining) algoritmus a keresési teret szűkíti és ezzel gyorsítja a szekvenciális minták megtalálását.

Mind a FreeSpam és a 7.7. részben bemutatandó algoritmus a probléma egy bővebb osztályára alkalmazható, ahol a sorozatok nem diszkrét elemekből állnak, hanem elemek véges részhalmazaiból. Mivel az én esetemben a web-statisztikákhoz nincs szükség erre az általánosításra, ezért én a továbbiakban az algoritmusok „butított"változatát közlöm, az eredetiek megtalálhatók a hivatkozott dokumentumokban.

Ha $\alpha = \langle s_1, s_2, \ldots, s_l \rangle$ egy sorozat, akkor az $s_1 \cup s_2 \cup \ldots \cup s_l$ halmazt *vetített elemhalmaznak*[22] hívjuk. Az algoritmus arra a megállapításra épül, hogy ha egy X halmaz nem gyakori, akkor egyetlen olyan sorozat sem lehet gyakori, aminek a vetített elemhalmazának X részhalmaza.

Legyen $f\_lista = \langle x_1, x_2, \ldots, x_n \rangle$ az összes gyakori elem az inputsorozat adatbázisban. Ezek alapján a sorozatok n halmazokba oszthatók a következő módon: az első halmazba az olyan sorozatok kerülnek, amelyek csak x_1-et tartalmazzák, a másodikba az olyanok, amelyek x_2-t igen, de egyetlen elemet sem tartalmaznak a $\{x_3, x_4, \ldots, x_n\}$ halmazból, és így tovább, a k-adik halmazba olyan sorozatok tartoznak, amelyek tartalmazzák x_k-t, de egyetlen elemet sem az $\{x_{K+1}, \ldots, x_n\}$ halmazból.

Az algoritmus alapmegállapítása szerint, amikor kiszámoljuk egy p sorozat vetített adatbázisát[23], akkor X gyakori elemhalmaza már ismert, ezért csak azokat az elemeket kell vetíteni. Ezzel nagyban csökkenti a megvizsgálandó elemek számát. Ezért az adatbázis vetítés a következő módon történik: az egész inputsorozat adatbázisból elhagyjuk azon elemeket, amik nincsenek benne X-ben, majd az így kapott sorozatokat vetítjük p-re. Így rekurzívan az algoritmus meg tudja találni a vetített adatbázisokat és az összes gyakori szekvenciális mintát.

sid	S
1	$a\,(abc)\,(ac)\,d\,(cf)$
2	$(ad)\,c\,(bc)\,(ae)$
3	$(ef)\,(ab)\,(df)\,cb$
4	$eg\,(af)\,cbc$

1.4. táblázat: Példa

[22] projected itemset
[23] projected database

3. Példa
Tekintsük a 1.4 példában lévő adatbázist. Legyen minsup 2. Ekkor az algoritmus az első lépésben kigyűjti az összes gyakori elemet (itt „elem:támogatottság" formában írom őket). Így a következőt kapjuk: $f\_lista = a:4, b:4, c:4, d:3, e:3, f:3$. *Ez máris 6 darab 1 hosszú szekvenciális mintát szolgáltat.*

Az f_lista alapján a szekvenciális minták 3 részhalmazba oszthatók:

- *azok, amik csak az a-t tartalmazzák,*
- *azok, amik tartalmazzák a b-t, de b utáni elemet nem,*
- *azok, amik tartalmazzák a c-t, de c utáni elemet nem, és így tovább*
- *valamint azok, amik tartalmazzák f-et.*

A további szekvenciális minta megtalálásához 6 vetített adatbázist tudunk konstruálni. A nem gyakori elemeket, mint esetünkben a g, ki lehet hagyni az adatbázisokból.

A vetített adatbázisok bányászása a következőképpen folytatódik:

- *Az a-vetített adatbázis a következő:* $\{aaa, aa, a, a\}$. *Erre rekurzíve alkalmazva az algoritmust, azaz kiszámolva az aa-vetített adatbázist:* $\{aa\}$ *megtaláljuk az* $aa:2$ *szekvenciális mintát. Ezen az ágon a következő szekvenciális mintákat találtuk:* $aa:2$.

- *A b-vetített adatbázis a következő:* $\{a\,(ab)\,a, aba, (ab)\,b, ab\}$. *Az így kapott adatbázisra rekurzíve alkalmazva az algoritmust a következő új mintákat tudjuk azonosítani:* $\{ab:4, ba:2, (ab):2, aba:2\}$.

- *A c vetített adatbázisa a következő:* $\{a\,(abc)\,(ac)\,c, ab\,(bc)\,a, (ab)\,cb, acbc\}$. *Ebből az adatbázisból a következő 2-mintákat lehet beazonosítani:* $\{ac:4, (bc):2, bc:3, cc:3, ca:2, cb:3\}$. *Rekurzíve folytatva az algoritmust az ac-vetített adatbázisát* $(\{a\,(abc)\,(ac)\,c, ac\,(bc)\,a, (ab)\,cb, acbc\})$ *bányászva a következő 3-mintákat lehet azonosítani:* $\{acb:3, acc:3, (ab)\,c:2, aca:2\}$. *Folytatva a rekurziót, ha az acb vetített adatbázisa:* $\{ac\,(bc)\,a, (ab)\,cb, acbc\}$, *ez nem generál új 4-mintákat. Így ezen az ágon a rekurzió befejeződik.*

- *A többi ág bányászása ugyanígy folyik, azaz mindig az előző szinten azonosított szekvenciális minták által meghatározott adatbázis minden elemére kiszámoljuk a vetített adatbázisokat, majd azokban keressük az előző szintnél 1-el hosszabb gyakori sorozatokat, majd az így megtaláltakat felvesszük a végső eredményt tartalmazó halmazba és rájuk is végrehajtjuk az algoritmust.*

Mint látszik az algoritmus azzal gyorsította a keresést, hogy az adatbázis vetítésekkel csökkenti a keresési teret, de amikor a k-adik szinten a k hosszú gyakori sorozatokat keressük, akkor az összes kombinációt ellenőrizni kell, ami nagyon költséges tud lenni, ahogy a sorozat hossza növekedik.

7.7. A PrefixSpan algoritmus[6]

Az algoritmus a FreeSpan továbbfejlesztése (ugyanazon szerzők által) a vetített adatbázis csökkentése és a minták rendezése irányába, amivel gyorsítható a futás és csökkenthető a memóriahasználat.

Az algoritmushoz szükség lesz egy-két definícióra.

3. Definíció
Tegyük fel, hogy az elemhalmazok rendezettek. Legyen $\alpha = e_1e_2 \dots e_n$, *ahol minden* e_i *gyakori elem ISA-ban. Egy* $\beta = e_1^{'}e_2^{'} \dots e_m^{'}, m \leq n$ *az* α prefixe *akkor és csak akkor, ha* $e_i^{'} = e_i, (i \leq m-1)$, $e_m^{'} \sqsubseteq e_m$ *és* $\left(e_m - e_m^{'}\right)$*-ben minden gyakori elem abc sorrendben* $e_m^{'}$ *utáni. Például: a, aa és* $a\,(abc)$ *prefixe* $a\,(abc)\,(ac)\,d\,(cf)$*-nek, de sem ab sem* $a\,(bc)$ *nem prefix, ha* $a\,(abc)$ *prefix minden eleme gyakori ISA-ban.*

4. Definíció
Legyen $\alpha = e_1e_2 \dots e_n$, *ahol minden* e_i *gyakori elem ISA-ban.* $\beta = e_1e_2 \dots e_{m-1}e_m^{'}, m \leq n$ *legyen az* α *prefixe.* $\gamma = e_m^{''}e_{m+1} \dots e_n$ *az* α suffixe β*-ra nézve, azaz* $\gamma = \alpha/\beta$, *ahol* $e_m^{''} = \left(e_m - e_m^{'}\right)$. *Másképpen írva* $\alpha = \beta\gamma$. *Például:* $s = a\,(abc)\,(ac)\,d\,(cf)$ *ara vonatkozó suffixe* $(abc)\,(ac)\,d\,(cf)$, *aa-ra vonatkozó suffixe* $(\_bc)\,(ac)\,d\,(cf)$

Ezen definíciók alapján a probléma rekurzíve dekomponálható. Ezt mondja ki a következő tétel.

3. Tétel

- *Legyen* $\{x_1, x_2, \dots, x_n\}$ *az* 1*-sorozatok halmaza ISA-ban. Az ISA-beli összes szekvenciális minta felbontható* n *részhalmazra, ahol az* i*-edik halmazbeli minták prefixe* x_i.

- *Legyen* α *egy* l*-minta,* $\{\beta_1, \beta_2, \dots, \beta_m\}$ *azon* $l+1$*-minták halmaza, amelyeknek prefixe* α. *Az összes* α *prefixű szekvenciális minta felosztható* m *darab részhalmazra, ahol az* i*-edik halmazbeli minták prefixe* β_i.

Továbbá még két definícióra és egy tételre van szükség az algoritmus kimondásához.

5. Definíció
Legyen $\alpha \in ISA$. α vetített adatbázisa $ISA|_\alpha$, *az ISA-bane levő* α*-ra vonatkozó suffixek halmaza.*

6. Definíció
Legyen $\alpha \in ISA$, *és* β *egy* α *prefixű sorozat.* β $ISA|_\alpha$ vetített adatbázisbeli támogatottsága *azaz* $támogatottság_{ISA|_\alpha}(\beta)$, *az* $ISA|_\alpha$ *beli* γ*-k száma, ahol* $\beta \sqsubseteq \alpha\gamma$.

4. Tétel
Legyen α *és* β *két folytonos minta ISA-ban úgy, hogy* α β *prefixe. Ekkor*

- $ISA|_\beta = (ISA|_\alpha)\,|_\beta$,

- $\forall\gamma$ *aminek* α *prefixe:* $támogatottság_I SA\,(\gamma) = támogatottság_{ISA|_\alpha}(\gamma)$,

- *α vetített adatbázisának a mérete nem haladhatja meg ISA méretét.*

Ezek alapján a PrefixSpan algoritmus működését a 1.4 segítségével szemléltetem. Tegyük fel, hogy $minsup = 2$.

Az algoritmus az első lépésben megkeresi az összes 1 hosszú szekvenciális mintát. Esetünkben ezek a $a:4, b:4, c:4, d:3, e:3, f:3$. Ezek alapján a keresési tér felosztható különálló részekre, ahol az elsőben az a prefixű sorozatok tartoznak, a másodikba a b prefixűek és így tovább.

Ezek után a megadott prefixeknek megfelelő vetített adatbázisokat kell megkonstruálni, majd azokat rekurzíve bányászni. A vetített adatbázis például az a esetében a következő: $(abc)\,(ac)\,d\,(cf)\,,(\_d)\,c\,(db)\,(ae)\,,(\_b)\,(df)\,cb,(\_f)\,cbc$. Ebben a vetített adatbázisban a gyakori elemek az $a:2, b:4, \_b:2, c:4, d:2$ és az $f:2$. Ezek alapján az 2 hosszú szekvenciális minták az $aa:2, ab:4, (ab):2, ac:4, ad:2, af:2$.

A rekurzió folytatódik tovább, most a fenti a prefixű minták által felosztott teret kell tovább-bányászni. Először az aa vetített adatbázisát kell meghatározni és bányászni, majd így tovább.

Belátható, hogy fent vázolt algoritmusnak több előnye is van az eddigiekkel szemben: sokkal kisebb keresési teret jár be, mivel mindig csak a már meglévő mintákat növeli; a sorozatos vetítések alapján a keresési tér folyamatosan szűkül, így a hosszú sorozatok sem lassítják le nagyon az algoritmust.

Az algoritmus legfőbb lassítója a vetített adatbázisok konstruálása, ez valamelyest gyorsítható pszeudóvetítés alkalmazásával, erről bővebben majd a BIDE algoritmusnál írok a 7.8. részben.

7.8. A BIDE algoritmus[8]

A BIDE algoritmus a CloSpan-hoz hasonlóan a zárt szekvenciális mintákat keresi. A CloSpan-nal ellentétben azonban nem tárolja az összes eddig megtalált jelöltet a memóriában, így nagyságrendekkel gyorsabb annál és az erőforrás nagyságrendekkel alacsonyabb.

Az algoritmus felteszi, hogy létezik egy lexikografikus rendezés az elemsorozatokon. Ezen rendezés segítségével fel lehet építeni egy fát, aminek a gyökere $\varnothing$-al van címkézve és egy $N:k$csúcsnak olyan gyerekei vannak, amik $Nl:m$ alakúak, ahol $l \in E$, és k valamint m az adott csúcs támogatottsága. Ha ebből a fából elhagyjuk a nem gyakori elemeket, akkor egy lexikografikusan rendezett gyakori fát kapunk.

Egy csúcs a fában tekinthető egy prefix sorozatnak, ahonnan a gyerekei egy E-beli l elem hozzáadásával keletkeznek. Ezen E-beli elemek között lehetnek nem gyakoriak is a prefixsorozatra nézve, így az ezekkel való kiterjesztéstől eltekinthetünk.

Az algoritmus alapja a kétirányú kiterjesztés (BI-Directional Extension). Az előrefelé való kiterjesztés növeli a prefix mintákat és egyben ellenőrzi a zártságot, míg a hátrafelé való kiterjesztés ellenőrzi a zártságot és szűkíti a keresési teret.

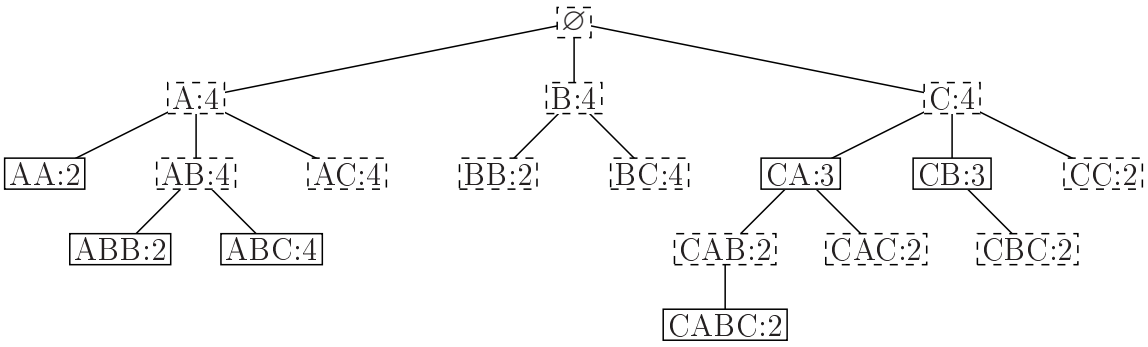
Először bemutatom a kétirányú kiterjesztés, a pszeudovetítés és a BackSpan algoritmust, majd ezek segítségével ismeretem magát a BIDE algoritmust.

A kétirányú kiterjesztés

Az eddigi algoritmusok úgy működtek, hogy legyártották a jelölteket, majd mindegyikre ellenőrizték, hogy tényleg zárt-e azaz, hogy nincs-e már egy olyan minta, ami miatt a jelölt nem lesz zárt. Ugyanígy azt is ellenőrizni kell, hogy az új jelölt nem zár-e ki egy már zártnak titulált mintát az előzőekből. Ezt a szakirodalom *sub-pattern checking-nek* és *super-pattern checking-nek* nevezi. Ezen állítás igazolására tekintsük az 1.5. példát. Ha itt a $B \leq A \leq C$ lexikografikus rendezést tekintjük akkor, amikor az $ABC : 4$ elemet vizsgáljuk, akkor az előzőleg már megtalált $BC : 4$ elem törölhető, mert elveszti a zártságát, de amikor a $C : 4$-et vizsgáljuk, akkor már van egy $ABC : 4$ jelöltünk, így a $C : 4$ nem lesz zárt. Ezzel a példával beláttuk, hogy szükség van mind a kétféle ellenőrzésre.

sid	S
1	CAABC
2	ABCB
3	CABC
4	ABBCA

1.5. táblázat: Példa



1.4. ábra: A 1.5. példában lévő adatbázis alapján generált fa. A szaggatott vonallal jelölt csúcsok nem zártak. A csúcsokban fel van tüntetve a támogatottság.

Azért, hogy ne kelljen az összes jelöltet a memóriában tárolni, egy új eljárást fejlesztettek ki. A zárt szekvenciális minták definíciója szerint, ha egy $S = e_1 e_2 \dots e_n$ sorozat nem zárt, akkor kell léteznie legalább egy e' elemnek, amivel a sorozatot ki lehet terjeszteni egy olyan S' sorozattá, aminek ugyanannyi a támogatottsága. Ezt a kiterjesztést 3 féle módon lehet elvégezni: 1. $S' = e_1 e_2 \dots e_n e'$ (előrefelé való kiterjesztés) 2. $\exists i \, (1 \leq i < n), S' = e_1 e_2 \dots e_i e' e_{i+1} \dots e_n$ (visszafele való kiterjesztés) 3. $S' = e' e_1 e_2 \dots e_n$ (szintén visszafele való kiterjesztés).

Következzen az egész algoritmus alapját képező tétel[24].

5. Tétel
Ha egy S prefix sorozathoz nincs olyan e' elem, amivel vagy előre vagy visszafele kiterjeszthető, akkor a sorozat zárt. Különben nyílt.

Látszik, hogy ezek alapján már csak azt kell ellenőrizni, hogy léteznek-e ilyen elemek. Az előrefelé való kiterjeszthetőséget könnyű ellenőrizni a következő tétel alapján.

6. Tétel
Egy S minta előrefelé való kiterjesztéshez használható elemek megegyeznek a lokálisan gyakori elemekkel, amiknek a támogatottsága egyenlő $sup\,(S)$-sel.

A hátrafele való kiterjesztéshez szükségünk lesz egy-két definícióra.

7. Definíció
Legyen S sorozat, ami tartalmaz egy $e_1e_2 \ldots e_i$ prefix i-sorozatot. A prefix sorozat utolsó előfordulásán[25] *az S elejétől az e_i utolsó előfordulásáig tartó részsorozatot értjük. Például: az $ABCABBC$ sorozatban az AB prefix utolsó előfordulása az $ABCABB$.*

8. Definíció
Legyen S sorozat, ami tartalmaz egy $S_p = e_1e_2 \ldots e_n$ prefix n-sorozatot. A prefix sorozat i-edik utolsó előfordulásán[26] *az LL_i-t értjük ahol, LL_i az utolsó e_i az S_p utolsó előfordulásában, ha $i = n$, illetve az utolsó olyan e_i az S_p utolsó előfordulásában, ahol még LL_i-nem meg kell előznie LL_{i+1}-et, ha $1 \leq i < n$. Például: az $ABCABBC$ sorozatban az AB 2. utolsó előfordulása az utolsó B betű, még az 1. utolsó előfordulása az utolsó előtti B betű.*

9. Definíció
Legyen S sorozat, ami tartalmaz egy $S_p = e_1e_2 \ldots e_n$ prefix n-sorozatot. A prefix sorozat i-edik maximum periódusán[27] *az $e_1e_2 \ldots e_{i-1}$ első előfordulása utántól az S_p i-edik utolsó előfordulásáig tartó részsorozatot értjük, ha $1 < i \leq n$, illetve az S_p 1. utolsó előfordulásáig tartó részsorozat, amennyiben $i = 1$. Például: a $BACABBC$ sorozatban az AB 2. maximum periódusa az CAB sorozat illetve, az 1. maximum periódusa az $\varnothing$.*

Ezek után a tétel:

7. Tétel
Legyen $S_p = e_1e_2 \ldots e_n$ egy prefix n-sorozat. Ha $\exists i\,(1 \leq i \leq n)$ és $\exists e'$, ami az S_p minden i-edik maximum periódusában szerepel, akkor és csak akkor e' előrefele való kiterjesztéshez használható elem. *Például, ha $S_p = AC : 4$-et a 1.5. példában szereplő sorozatokra nézzük, akkor S_p 2. maximum periódusa $CAABC$-re AB, $ABCB$-re BCB, $CABC$-re B valamint $ABBCA$-ra BB, így látható, hogy B mindegyik-ben előfordul, így $AC : 4$ nem lehet zárt. De ha például $ABC : 4$-re nem találunk ilyen elemeket, így az zárt.*

[24] A tételek bizonyítása megtalálható az eredeti dokumentumban
[25] Last instance of a prefix sequence
[26] The i-th last-in-last appearance with regards to a prefix sequence
[27] The i-th maximum period of a prefix sequence

Ezek alapján már el tudjuk végezni a kétirányú kiterjesztést.

A pszeudovetítés

A lokálisan gyakori elemek megkereséséhez *pszeudóvetítést*[28] használja, hasonlóan a FreeSpan és PrefixSpan algoritmushoz. A pszeudóvetítés annyival tér el a valódi vetítéstől, hogy csak a mutatókat tárolja, amik az adott vetített sorozatok elejére mutatnak. Így nem kell ténylegesen legyártani a vetített adatbázist, ezzel memóriát takarítunk meg (és ha a memóriafoglalást is nézzük, akkor időt is).

A pszeudovetítés megértéséhez szükséges a következő pár definíció.

10. Definíció
Legyen S egy sorozat, ami tartalmaz egy prefix 1-sorozatot e_1-et. S elejétől e_1 első előfordulásáig tartó sorozatot az e_1 prefix 1-sorozat első előfordulásának[29] *nevezzük. Hasonló módon, rekurzíve definiálhatjuk az $e_1e_2e_3 \dots e_i$* utáni első előfordulását az $e_1e_2e_3 \dots e_ie_{i+1}$ $(i+1)$-sorozatnak, *miszerint az S elejétől az első olyan e_{i+1} elemig tartó sorozat, amit $e_1e_2e_3 \dots e_i$ megelőz. Például, az AB prefix sorozat első előfordulása a $CAABC$ sorozatban a $CAAB$ sorozat.*

11. Definíció
Ha S egy sorozat, ami tartalmaz egy $e_1e_2e_3 \dots e_i$ prefix i-sorozatot, akkor S ezen i-sorozat utáni részét az $e_1e_2e_3 \dots e_i$ vetített sorozatának[30] *hívjuk. Például, AB vetített sorozata $ABBCA$-ra a BCA sorozat.*

12. Definíció
A ISA input sorozat adatbázisnak az $e_1e_2e_3 \dots e_i$ sorozatra vetített részét, az $e_1e_2e_3 \dots e_i$ vetített adatbázisának[31] *hívjuk. Például az $\{CAABC, ABCB, CABC, ABBCA\}$ input-sorozat adatbázis AB-ra vetített adatbázisa $\{C, CB, C, BCA\}$.*

A fenti definíciók alapján az előzőekben leírt pszeudóvetítés működik. Valamint a legyártott mutatók segítségével már könnyen meg tudjuk járni a vetített adatbázist és meg tudjuk állapítani, mik a gyakori elemek és ezzel tudjuk növelni a fát. Ez az előrefelé való kiterjesztés fázisa. A 12. definícióban látható példára, ha $minsup = 2$, akkor gyakori elemek a $C:4$ és a $B:2$.

A BackScan algoritmus

Ezen algoritmus nélkül is működik a BIDE, de a használata nagyságrendekkel gyorsítja a futást és csökkenti a felhasznált memóriát.

Az 1.4. ábrán látható, hogy a $B:4$ ág nem adott egyetlen zárt prefix-et sem. Ha ezeket tudnánk detektálni, akkor nem kellene őket fölöslegesen növelni, így csökkenthetnénk a keresési teret. Újból szükségünk van egy-két definícióra.

[28] pseudo projection
[29] First instance of a prefix sequence
[30] Projected sequence of a prefix sequence
[31] Projected database of a prefix sequence

13. Definíció
Ha az S sorozat tartalmaz egy $S_p = e_1e_2 \dots e_n$ prefix n-sorozatot, akkor a LF_i az S_p elsőben i-edik utolsó előfordulása[32] *az e_i utolsó előfordulása az S_p első előfordulásában, ha $i = n$, illetve az e_i utolsó előfordulása az S_p első előfordulásában, ahol még LF_i-nek meg kell előznie LF_{i+1}-et, amennyiben $1 \leq i < n$. Például: $S = CAABC$ és $S_p = CA$ esetén S_p 2. elsőben az utolsó előfordulása, az első A az S sorozatban.*

14. Definíció
Legyen S sorozat, ami tartalmaz egy $S_p = e_1e_2 \dots e_n$ prefix n-sorozatot. A prefix sorozat i-edik félig maximum periódusán[33] *az $e_1e_2 \dots e_{i-1}$ első előfordulása utántól az S_p elsőben az i-edik utolsó előfordulásáig tartó sorozat, ha $1 < i \leq n$, illetve S kezdetétől az S_p elsőben az 1. utolsó előfordulásáig tartó sorozat, amennyiben $i = n$. Például: $S = ABCB$ és $S_p = AC$ esetén a S_p 2. félig maximum periódusa B, illetve 1. félig maximum periódusa $\varnothing$.*

Ezek alapján a tétel a következő.

8. Tétel
Legyen $S_p = e_1e_2 \dots e_n$ egy prefix n-sorozat. Ha $\exists i\,(1 \leq i \leq n)$ és $\exists e'$, ami az S_p összes i-edik félig maximum periódusában előfordul, akkor az S_p utáni részfát le lehet vágni.

A $B : 4$ esetében ez az elem az A, így nyugodtan be lehet fejezni ennek az ágnak a kiterjesztését.

A BIDE algoritmus

Első lépésként az algoritmus megkeresi az adatbázisban az 1-sorozatokat és megkonstruálja azok vetített adatbázisát. Ezek után minden 1-sorozatot prefix-nek tekint és ellenőrzi, a *BackScan* algoritmussal, hogy az adott ágon kell-e tovább növelni a fát, vagy az adott ágat le lehet vágni[34]. Amennyiben nem lehet levágni az adott ágat, úgy meghatározza a visszafele való kiterjesztéshez használható elemeket és meghívja a rekurzív függvényét a vetített adatbázissal, a prefix sorozattal, a *minsup*-pal és az előbb meghatározott elemekkel.

A függvény az átadott prefix vetített adatbázisára meghatározza a lokális gyakori elemeket. Majd kiszámolja az előrefelé való kiterjesztéshez használható elemeket. Amennyiben nincs egy előrefelé és visszafele kiterjesztéshez használható elem sem, akkor az átadott prefixsorozat zárt. Ezek után a prefix-et sorban az összes lokálisan gyakori elemmel konkatenálja (lexikografikus rendezés szerint), majd az így kapott új prefix-eknek meghatározza a vetített adatbázisát, majd amennyiben ezek segítségével a prefix-et jelképező ágat nem lehet levágni a BackScan algoritmus segítségével, akkor kiszámolja a visszafele való kiterjesztéshez használható elemeket és meghívja a rekurzív önmagát.

A fenti algoritmusban említett BackScan elhagyható, anélkül is működik a zárt prefixek meghatározása, de annak megléte nagy *minsup* esetén (≥ 0.025) nagyságrendekkel gyorsítja a futást és csökkenti a memóriahasználtságot.

[32] The i-th last-in-first appearance w.r.t. a prefix sequence
[33] The i-th semi-maximum period of a prefix sequence
[34] A vágás létjogosultságáról és az algoritmusról majd később lesz bővebben szó.

7.9. Összefoglalás

Az előzőekben bemutattam több algoritmust is, amik a gyakori mintabányászat feladatát oldják meg. Az algoritmusokat történelmi sorrendben prezentáltam.

Kezdtem az 1994-ben bemutatott Apriori algoritmussal[5], ami az asszociációs szabályokra épül és a jelöltgenerálás majd ellenőrzés elvén működik. Majd a A GSP algoritmust[7] mutattam be, amit ugyanaz a szerzőpáros, Srikant és Agrawal, alkotott 1997-ben, akik az Apriori algoritmust. A GPS több megszorzítást alkalmaz, mint az Apriori algoritmus, úgy mint a minimális- és maximális-rés, és a csúszóablak méret. Az algoritmus bár nagyságrendekkel gyorsabb, mint az Apriori algoritmus, még mindig a jelöltgenerálás majd ellenőrzés elvén működik. Az előrelépés mindössze annyi, hogy a jelöltgenerálást jobban végzi, ezért nem keletkezik annyi fals jelölt.

Az első nagy előrelépés a A WAP-mine algoritmus[1] volt, ami már egy merőben új irányból közelítette meg a problémát, bevezette a WAP-fa adatstruktúrát, amit arra használt, hogy az inputsorozatokat helytakarékosan tárolja valamint, hogy a gyakori mintákat annak segítségével könnyebben tudja legenerálni. Szintén itt került bevezetésre a prefixsorozat fogalma is. Az egész új megközelítés arra a triviális megállapításra épül, miszerint ha e egy gyakori elem a P-re vonatkozó prefixek között az ISA-ban, akkor eP egy szekvenciális minta ISA-ban. A WAP-mine algoritmus nagyságrendekkel gyorsabb, mint az őt megelőző bármelyik algoritmus és az új adatstruktúra miatt a memóriafelhasználása is kisebb.

Következett a FreeSpan algoritmus[6], ami bevezette a vetített elemhalmaz fogalmát, és arra a megállapításra épül, hogy ha egy X halmaz nem gyakori, akkor egyetlen olyan sorozat sem lehet gyakori, aminek a vetített elemhalmazának X részhalmaza. A vetített adatbázisokat felhasználva az algoritmus eddig nem látott módon szűkíti a keresési teret és ezzel teljesítményben felülmúlja az eddigi próbálkozásokat. Az algoritmus egy továbbfejlesztett változatát is bemutattam, az a PrefixSpan algoritmus[6]. Ez a vetített adatbázisok készítését javította fel, így még tovább szűkítette a keresési teret, valamint bevezette a pszeudovetítést, mint memóriahasználtság csökkentő eljárást.

Végül bemutattam a BIDE algoritmust[8], ami már zárt szekvenciális minták keresésére lett kifejlesztve, és bonyolult kétirányú kiterjesztés segítségével úgy keresi meg azokat, hogy közben nem kell az addig megtalált jelölteket a memóriában tartania. Azzal, hogy nem kell a memóriában tárolnia az össze jelöltet, nagyban felülmúlta az elődeit, és még a bonyolultabb feladatot (zárt minta keresést) is gyorsabban végzi, és közben kevesebb memóriát használ, mint a többiek a nyílt minták keresését.

Jól megfigyelhető, hogy a problémakörben hogyan fejlődnek az algoritmusok a kezdetben még nem ide kitalált, de a problémára ráerőszakolt megoldástól az egyre specifikusabb és bonyolultabb algoritmusok felé. Az elmúlt, alig több mint 10 év alatt az ezirányú fejlődést nagyban ösztönözte a web-statisztika készítés, és ezen belül a felhasználói viselkedésmodellezés kihívása. Ez nagyon jól látszik a téma egyre növekvő irodalmán és a számtalan különféle, egymást lekörözö algoritmuson, amik ezen rövid idő alatt megjelentek. Én itt csak a legfontosabb mérföldkőnek számító algoritmusokat mutattam be, ezen kívül még nagyon sok létezik, mint például az A-Close, CLOSET, CHARM, CLOSET+ és a CloSpan.

Ahogy a a BIDE algoritmussal én is érintettem, a kutatások egyre inkább a zárt szekvenciális minták generálása felé mozdultak, mivel azok sokkal kompaktabban fejezik ki a kívánt információt, esetünkben a felhasználó által bejárt útvonalakat.

8. Visszatekintés

Ebben a fejezetben röviden áttekintettem az internetes tartalom fejlődését, kitérve az egyes főbb változások mögött meghúzódó okokra is. Majd bemutattam a mai internetes világot meghatározó web-es technikákat, úgy mint az AJAX-ot a CSS-t és a sablonos oldalfelépítés módszerét (ez utóbbi azért, mert a következő fejezetben bemutatott, általam készített web-statisztika gyártó programot web-es felületen valósítottam meg és célom volt azt a mai kor követelményeinek megfelelően megvalósítani).

Majd kitértem a statisztika készítéshez szükséges adatok gyűjtésének különféle módszereire és magukra a figyelt értékek mibenlétére is. Az adatgyűjtési eljárások bemutatásánál kitértem arra, hogy a különféle modern web-es technikák, mint például az AJAX, milyen hatással vannak azokra.

Majd az utolsó nagy alfejezetben bemutattam a szekvenciális minta bányászat problémakörét, amit a web-statisztika készítésnél a gyakori felhasználói útvonalak beazonosításához használnak. Itt a bemutatott algoritmusokon keresztül bemutattam a témakör történelmét a kezdetektől napjainkig. Minden nagyobb mérföldkőnek számító algoritmus részletesen ismertettem, a hozzájuk szükséges teljes matematikai háttérrel.

2. fejezet

Egy tényleges megvalósítás elemzése

1. A cél

A mai világban sokféle statisztikát készítő programot illetve web-oldalt lehet találni az interneten. Ezek között vannak bonyolultabbak és egyszerűbbek is. Olyan azonban nem nagyon akad,[1] ami nem csak a számadatokat mutatná meg az embernek, hanem a következtetéseket is levonná belőle. Gondolok itt az olyan kijelentésekre, hogy

- „az oldal látogatottsága növekszik”
- „az *xy* oldalra kikerült a hivatkozásunk”
- „a látogatók általában az *ab - cd -ef* illetve az *ab-gh* útvonalakat járják be az oldalon”[2]

pedig a hétköznapi embernek pont erre lenne szüksége. Hiába rajzolnak neki egy cikkcakkos vonalat, nem biztos, hogy azt fejben tudja simítani és megállapítani, hogy most az hosszú távon nő vagy csökken.

Pusztán a számok és a linkek statisztikái alapján nem tudja megállapítani, hogy jó volt-e ezt vagy azt a hivatkozást a kezdőlap közepére kitenni, de ha látja, hogy a látogatók a kezdőlapról indulva az adott linket érintő útvonalat járnak be, akkor el tudja dönteni, hogy jó volt-e a döntés vagy sem. Bár léteznek már ilyen software-ek, de általában fizetősek és nem létezik online szolgáltatásként vagy csak a kutatásokat reprezentáló, nem a nagyközönség számára készült verzióik léteznek.

Persze fontos az is, hogy aki akar, az láthasson számsorozatokat, és minden egyéb szokásos kimutatást, de lehetőséget kell adni a fent vázolt intelligensebb és egyszerűbb jelentés megnézésére is.

[1] Itt csak az általam ismert web-statisztikát gyártó oldalak alapján tudok következtetést levonni. Továbbá az irodalmazás alatt is arra a megállapításra jutottam, hogy nem nagyon vannak olyan programok, amik az általam legfontosabbnak tartott felhasználói viselkedés modellezést tartalmazzák, illetve csak *off-line* módban elérhető kereskedelmi programok léteznek erre a problémára.

[2] Ezt a témakört *sequential parretn minig* néven ismeri az angol nyelvű szakirodalom. Erről még lesz később bővebben szó.

Nekem itt az a célom, hogy bemutassam, hogyan lehet ezt a statisztikát gyártó web-oldalt megvalósítani. Milyen háttérismeretek kellenek ehhez, milyen algoritmusok, illetve, hogy ezen kívül bemutassam, hogyan kell egy, a nagyközönségnek szánt web-oldalt felépíteni, ami ezt szolgáltatja. Mindezt úgy, hogy szem előtt tartom az első fejezetben bemutatott modern web-fejlesztési irányelveket.

További cél lehetne, hogy egy olyan portálmotor létrehozása, ami az itt készített statisztikák és megfigyelések alapján dinamikusan alakítja a web-oldalt az adott felhasználó szokásaihoz igazítva. Mivel a megfigyelések alapján, például a bejárt útvonalak és a legyakoribb tartalom segítségével sokkal személyreszabottabban lehetne a tartalommal a látogatókat megcélozni. Ha megfigyeltük, hogy valami a kezdőoldalról általában felkeres egy meghatározott oldalt, akkor akár lehet célunk, hogy azt neki könnyen elérhetővé tegyük és kirakjuk a nyitólapra, de akár az is, hogy az odafele lévő útvonalakon célzott hirdetés jelenítünk meg.

Ehez a statisztikakészítő oldal és a portálmotor között egy speciális protokollt kéne definiálni amivel tudnak egymással kommunikálni és persze meg kéne írni a megfelelő protálmotort. Ezen feladatok túlmutatnak a dolgozatom keretein, de tudni kell, hogy a világ ebbe az irányba halad. Nem tudok róla, hogy létezne ilyen szabadon vagy kereskedelmi forgalomban kapható termék, de egyes cégek már alkalmazzák a web-oldalak ilyen módon való hozzánk idomítását.

2. A program áttekintése

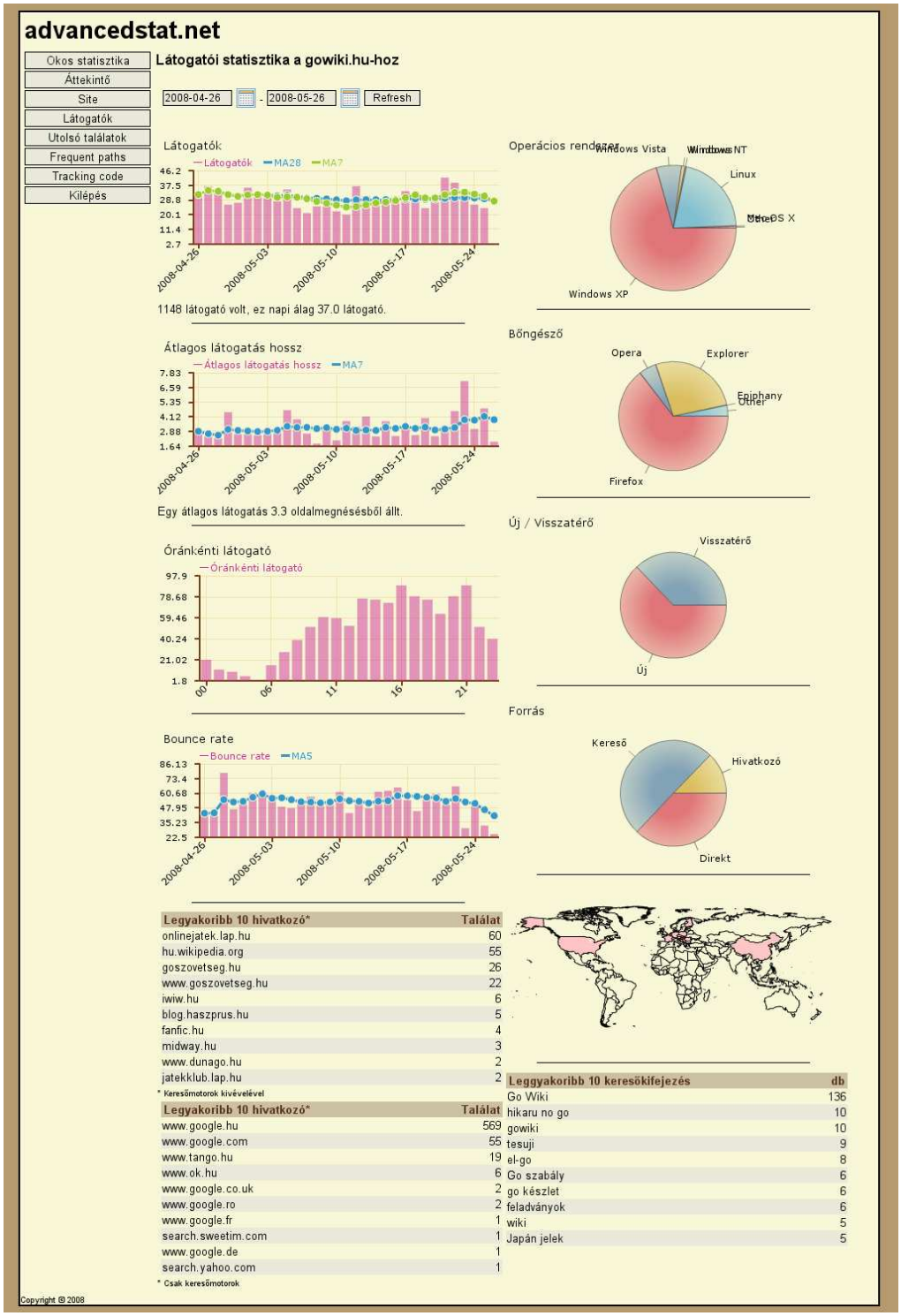
A web-oldalon belépve a 2.1. ábrán látható képernyőkép fogadja a felhasználót. Az egyes kategorizált kimutatásokat a baloldali menüből lehet elérni. Ezek a következők:

Okos statisztika Azen az aloldalon a következő adatokat kapja az ember: rövid- illetve hosszútávu trend, az elmúlt heti látogatószám, napi átlaglátogató illetve az átlagos látogatáshossz, a megszűnt, ujraéledő és aktív és az új hivatkozások (az aktív hovatkozásoknál találati szám is). Minndezeket az adatokat folyószöveg kíséri.

Áttekintő A vizuálisabb egyének kedvéért ezen az oldalon a megszokott grafikonok vannak, a napi látogató illetve találati számmal illetve a havi látogatószámmal. Minden grafikonon 7 illetve 28 napos mozgó átlag is fel van tüntetve.

Web-oldal Ezen az oldalon a legelső 10 belépő-, kilépő, ugró-, illetve legnépszerűbb oldal listája látható. Itt már az eddigiektől eltérően állítható a kezdő illetve a végső dátum a kimutatáshoz.

Látogatók A leg zsúfoltabb aloldal, itt szintén állítható dátumhatárral a következők láthatóak: 4 grafikon, az látogatószámmal, átlagos látogatáshosszal, ugrási rátával és óránkénti látogatószámmal. Az első 3-hoz mozgóátlag is tartozik. Ezenfelül van egy nagyítható sematikus világtérkép a látogatók ország szerinti elolszálával. Továbbá négy tortagrafikon, a operációs rendszerek és böngészők megoszlásáról, a látogatás forrásáról (kereső, direkt vagy hivatkozó általi) és a visszatérő illetve új lűtogatók



2.1. ábra: A program egy képernyőképe

megoszlásával. Valamint 3 táblázat a legjobb 10 hivatkozóval, keresőmotorral illetve a keresőkifejezésekkel.

Utolsó találatok Az tulsó pár (legördülö menüböl választható, hogy 5, 20, vagy 50) találat következő adatai: dátum, idő, osztnév, ország, operációs rendszer, böngésző, visszatérő vagy új látogató illetve hvatkozó és URL páros.

Gyakori utak Itt a felhasznákók által az utlsó 1 hétben legyakrabban bejárt útjait mutatja meg a program.

Követő kód A web-oldalba beillesztendő HTML részletet mutatja meg itt a program.

Beállítások Ezen az oldalon változtathatjuk meg a jelszavunkat, az oldal nyelvét illetve az automatikusan újratöltődő oldalrészeknél az újratöltés gyakoriságát.

Az áttekintő oldal illetve az utolsó látogatókat bemutató oldal a háttérben 5 percenként frissül, az utolsó frissítés időpontjáról az oldal alján lévő időbéjegy tájékoztatja a felhasználót.

Amennyiben a statisztika tulajdonos valmi oknálfogva nem tud belépni az oldalra kérhet jelszóemlékeztetőt a felhasználóneve és az e-mail címe megadásával. Amennyiben ezeket helyesen adja meg a rendszer egy e-mail-t küld neki amiben egy speciális egyszerhasználatos URL van amire kattintva be tud lépni és megtudja változtatni a jelszavát.

Belépésnél az „Emlékezz rám” jelöllőnégyzettel kérhetjül a belépési adatok eltárolását, hogy a következő oldalfelkereséskor autómatikusan beléptessen minket a rendszer.

3. Egy web-statisztika gyártó web-oldal anatómiája

Az alkalmazás három elkülöníthető részre bontható, ezek közül az egyik az adatbázis, amit a másik kettő használ. A fennmaradó két komponens: a számláló és a jelentés-generáló. Ez utóbbi kettő nem kapcsolódik egymáshoz, csak a közösen használt adatbázis által. Egymástól függetlenül is működhetnek.

Számláló Ez a rész nem csinál mást, mint a látogatási adatokat az adatbázisba írja. Bővebben az itt alkalmazható technikákról a 6.1. részben lesz szó.

Jelentés-generáló Ez végzi az adatok felhasználónak való megjelenítéshez szükséges szűrését, kombinálását illetve formázását. Továbbá ez a rész hozza meg az esetleges „intelligens” következtetéseket is.

Meg kell jegyezni, hogy a jelentés-generáló bármilyen megfelelő formátumban lévő adaton működik, nem kell, hogy azt a számláló készítette, egyéb web-szerver naplókat konvertáló program által is feltölthető adattal. Ez utóbbi esetben, a megfelelő feltöltő és konvertáló mechanizmussal működhet egyfajta web-log-file kiértékelő szolgáltatásként is. Bár mint majd a 6.1. részben bemutatom, a számláló rész bizonyos esetekben több adathoz képes jutni, mint egy egyszerű web-szerver log-file.

A web-oldal tartalmaz továbbá még egy-két szorosan nem a statisztika-gyártáshoz kapcsolódó, de elengedhetetlen részt, ezek a következők:

- nyitóoldal (belépés, leírás, „körút”, „elfelejtettem a jelszavam”)
- regisztrációs oldal
- kategória böngésző és top lista

Ezeknél az oldalaknál is többféle technológiai csínyt alkalmazok. Ilyenek a regisztráció közbeni JavaScript-es, AJAX-os adat ellenőrzés, a robotok általi regisztrációt megnehezítő „olvasd el a szöveget a képen” megoldás. A többnyelvűséget elősegítő nyelvi file-ok használata, és persze a programozást megkönnyítő template-es megoldás a HTML és php részek szétválasztására. Valamint a statisztikát mutató résznél is felhasználom az AJAX-os megoldásokat a gyorsabb és gördülékenyebb megjelenítés elősegítéséért.

4. A fejlesztői környezet

A programot (web-oldalt) php nyelven írtam, a fejlesztés alatt használt változat az 5.2.3-1ubuntu6.3 volt, de elvileg minden 5-ös verzió számú fölött működik. Az adatbázis háttért egy 5.0.44 verziójú MySQL szerver biztosította, itt is minden 5-ös feletti verziónál elvileg működőképes a program. A további külső programkomponensek, amiket csatoltam is a megvalósításhoz, a következők: Smarty-2.6.19 (a jobb áttekinthetőség és tovább fejleszthetőség miatti template-es web-oldal megvalósításhoz), JPgraph-2.3 (egyes grafikonok megjelenítéséhez illetve az *anti-spam* szövegekhez, Open flash chart 1.9.7 (flash alapú grafikonok megjelenítéséhez) valamint a DIY Map-ot (a térkép megjelenítéséhez). Fontos szempont volt, hogy a teljes program csak nyílt forrású komponensekből álljon, egyrészt, mert annak elkötelezett hívője vagyok, másrészt, mert nem akartam vagyonokat költeni a diplomamunkámra.

5. Programozási megoldások

5.1. Az adatgyűjtés

Az én esetemben az első fejezetben bemutatott adatgyűjtési megoldások közül csak a JavaScript-es jöhetett szóba, de az is csak kiegészítő adatok gyűjtésére, mivel a felhasználók letilthatják a JavaScript-ek futtatását, így arra nem lehet teljes egészében támaszkodni. Előnye, hogy csak a web-oldalak betöltéskor keletkeznek adatok, sőt azok közül is csak azoknál, ahol az oldal üzemeltetője azt akarja, ezzel akár szűkítheti is a monitorozandó lapok halmazát.

A következő kis részlet kerül be a weboldalakba:

```
<script type="text/javascript" language="JavaScript"
src="http://stat/?p=h&m=g&id=1234"></script>
```

```
<noscript><img src="http://stat/?p=h&m=c&id=1234"
width="0" height="0"></noscript>
```

ahol `http://stat` a statisztikát gyűjtő web-szerver elérése, illetve `1234` a regisztrált felhasználó azonosítója. A web-oldal üzemeltetőjének ezt a kis részt kell minden web-lapnak az aljába illesztenie.

Mint látszik, van egy `<noscript>...</noscript>` rész, ami egy 0x0 felbontású képet jelenít meg (illetve nem jelenít meg, de lekéri a szervertől), és az így generált lekérés alapján jegyezzük be a naplóba azt, hogy a weboldalt meglátogatták.

Feltehető a kérdés, hogy vajon miért van szükség a JavaScript-es részre? Azért, mert ilyen esetben nem tudjuk megmondani, hogy az adott oldalra honnan kattintott a felhasználó és cookie-s felhasználó azonosítást sem tudunk elvégezni.

A script által beillesztett részlet nagyon hosszú lenne, ezért csak az ott futó program működését írom itt le. Először is a script lekéri a felhasználó azonosítására használt cookie egyedi sorszámát[3] Ha ezt nem kapja meg, akkor megpróbál egyet beállítani és aztán újra lekérni annak az értékét. Az újra lekérésre azért van szükség, hogy meg tudjuk állapítani, hogy sikerült-e a beállítás. Beilleszt a web-oldalba egy, a fenti `<noscript>...</noscript>`-es részhez hasonló képre való hivatkozást annyi eltéréssel, hogy most a böngészőtől lekéri a `document.referer`[4] értékét és ezt hozzáírja az URL-hez, a cookie-val egyetemben, amennyiben az létezik. Valamint egy 6 karakter hosszú véletlensorozatot is hozzácsatol az URL-hez, ezzel védve ki a proxy szerverek által való újraküldést illetve a *vissza* gomb memóriából való betöltése általi hivatkozás-kimaradást. Mivel így mindig különböző képre lesz a hivatkozás és így a böngészők azt mindig lekérik és ezzel generálják a naplózható lekérdezést.

Mint látszik a JavaScript-es megoldás sokkal több adatot képes szolgáltatni, de annak használata nélkül is boldogulni kell tudni. Anélkül egyetlen másik módszer sem képes érzékelni a felhasználói visszalépést, illetve a proxy szerverek meglétét, mint ahogy egyetlen másik módszer sem képes annyira pontosan követni a felhasználókat, mint a cookie-k használata.

Az első részben írtam róla, hogy az AJAX-os megoldások megnehezíthetik a web-statisztika gyártást. Sajnos ez ellen nem nagyon sokat lehet tenni a készítő oldalon, csak a web-oldal szerkesztője tud annyival segíteni a dolgon, hogy nem csak a statikus oldalakba teszi be a számláló kódot, hanem azon program által generált AJAX-os tartalmakba is, amik különálló oldalt alkotnak. Így azok számlálása is zavartalanul folyhat.

5.2. A cookie dilemma avagy hogyan azonosítsuk a látogatókat

Érdekes kérdés, hogy mikor tekintsünk két látogatót azonosnak a következő adatok felhasználásával: cookie, host, agent. Valamint, hogy az egyes alternatívák milyen következ-

[3] Ez egy véletlenszerűen generált 32 hosszú karaktersorozat.

[4] Azért van szükség erre a *referer trükkre*, mivel a HTTP_REFERER szerver változó értéke ilyenkor annak az oldalnak a címét adná vissza, ahova a számláló „kép"- be van illesztve, de nekünk nem arra van szükségünk, hanem arra, ahonnan az adott web-oldalra kattintott a látogató. Ezt csak ilyen módon tudjuk megszerezni. Ezért ehhez még akkor is szükség lenne JavaScript-es trükközésre, ha másért (cookie-ért) nem is akarnánk azt használni.

ményekkel járnak.

Logikus válasz lehet, hogy ha a cookie azonos, akkor a két felhasználó ugyanaz, ha esetleg más a host-neve, akkor dinamikus IP-t használ (vagy laptopot). A különféle döntések több problémát is felvetnek.

Ha csak 1 sort tartunk fel neki a `visitors` táblában, akkor mi legyen a `host` és az `agent` mezővel? Frissítsük, tartsuk meg az első értéket? A statisztikát néző ember kíváncsi lehet az IP-re (az összes IP-jére), sőt az ország beazonosítása is ez alapján történik.

Ha több sort adunk neki, akkor a `visits` táblában már két különböző szám fog megjelenni a látogatónál, pedig ugyanarról az emberről van szó[5].

Én az utóbbi megoldást választottam, mivel ez esetben kicsit bonyolultabb lekérdezésekkel, de elérhetők ugyanazok az eredmények, amit az első megoldás mellett tudnánk kapni, de fordítva ez nem lenne igaz.

5.3. Az objektumok és feladataik

A program a mostanában igen népszerű objektumközpontú szemléletben készült. Az egyes részek külön osztályként vannak megvalósítva:

`database` Egy közös átjárófelületet biztosít az adatbázis felé. Előnye, hogy ha más környezetbe helyezik át a programot és ott is SQL kompatibilis adatbázis van, de ha esetleg nem MySQL, akkor elég csak ezt az egy osztályt átírni és máris használható az alkalmazás. Ez az osztály végzi az adatbázishoz való csatlakozást és a lekérdezések eredményeit egy asszociatív tömbben adja vissza, ezzel nagyban egyszerűsíti a többi osztályban az adatbázis-műveletek elvégzését.

`hit` A tényleges számlálást végző osztály, ez generálja a monitorozandó oldalnak küldött JavaScript részt is.

`initdatabase` Az adatbázis kiürítését és felépítését végző osztály, a tényleges használat közben nincs szerepe sőt, a véletlen adatvesztések miatt a kódból ki is van kommentezve az ezt tartalmazó rész.

`iptocountry` Az `initdatabase` osztályhoz hasonlóan ennek is csak adminisztrátori szerepe van. Ez az osztály tölti le az internetről az IP-cím ország összerendeléseket tartalmazó adatbázist és aztán írja be a megfelelő adatbázistáblába.

`main` A nyitóoldal megjelenítéséért felelős osztály.

`otp` Az ún. *One Time Password* azaz az egyszer használatos jelszavas beléptetést és jelszóváltoztatást végző osztály.

`register` A leendő felhasználóknak a regisztrálását megvalósító programrész.

`helper` Különféle, több osztály által használt függvényeket tartalmaz.

[5] 32 hosszú véletlenszerű karakter-sorozatot tartalmazó cookie esetén majdnem biztos

`report` Az egyik legfontosabb osztály, ami a tényleges kimutatások megjelenítését végzi. Nagymértékben támaszkodik a `Helper` osztályra.

`chart` Ezen osztály metódusai állítják elő a különféle grafikonok és táblázatok adatait is. Nagymértékben támaszkodik a `Helper` osztályra.

`lang` Ez az osztály olvassa be a nyelvi file-t, majd szolgáltatja a tartalmát php-n belűl. Főként a `chart` osztály támaszkodik rá.

`settings` Ezen osztályon keresztül valósítja meg a web-lap a felhasználói adatok módosítását. Támaszkodik a `settings` és a `lang` osztályokra.

`prefixspan` Az általam megvalósított szekvenciális mintabányászatot végző osztály. A működéséről bővebben majd még lesz szó.

`progressbar` A `prefixspan` osztály alkalmazza az állapota megjelenztésére.

`trackingcode` A web-oldalba beillesztendő személyreszabott web-bug-ot állítja elő.

`script` A JavaScript-eket létrehozó osztály.

A fenti osztályokhoz több *template* (sablon) is tartozik, amik az éppen szükséges HTML oldal keretét adják. Ilyenek például a regisztrálás alatti lépések, illetve a különféle kimutatások aloldalainak a sablonjai.

Majdnem minden osztálynak van egy `display()` függvénye, ami a megjelenítésért felelős, azaz a böngésző által átadott adatok alapján a Smarty sablonok segítségével HTML forrást állít elő.

Az osztályokat egy `index.php` főprogram fogja össze, ami a különféle aloldalakért felelős osztályok példányosításán és azok `display()` függvényének a meghívásán túl inicializálja az adatbázist, a template-ekért felelős segédosztályt, és elvégzi a felhasználói ki- illetve beléptetést.

5.4. Érdekesebb programrészek

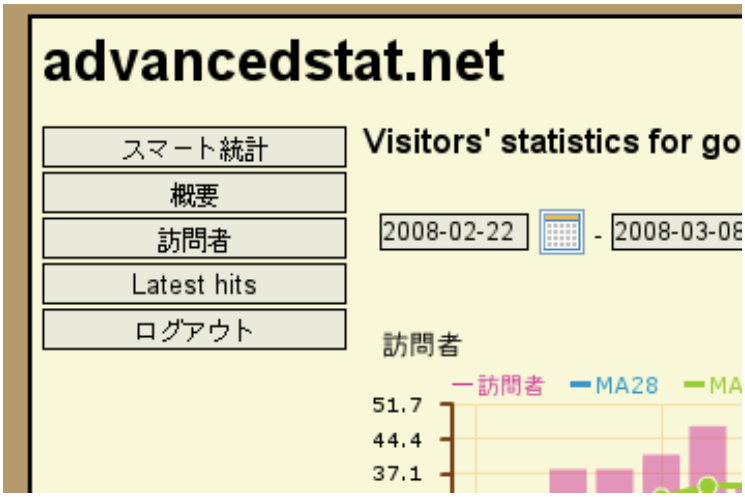
A beléptetés

A weboldal támogatja a normál jelszavas és a *cookie* általi azonosítást is. Ebből a felhasználó kívülről annyit lát, hogy a weboldalon a belépésnél lehetősége van egy pipával bejelölni, hogy az oldal emlékezzen rá. A háttérben az annyit tesz, hogy ebben az esetben a sikeres belépés után eltároltatjuk a böngészővel (ha támogatja a cookie-kat) a felhasználói azonosítót és az md5 függvény által titkosított jelszavat. Majd ha a felhasználó újra meglátogatja az oldalt és még nincs belépve (ezt az adott felhasználóhoz tartozó szerveroldali ún. *session* változók segítségével tároljuk, akkor ellenőrizzük, hogy van-e a böngészőben beállítva cookie és ha igen, akkor annak alapján kíséreljük meg a beléptetést[6].

[6]Azért írtam, hogy „kíséreljük meg", mert semmi sem biztosítja, hogy a cookie-kban megfelelő felhasználónév jelszó páros található. Ezen adatokat bárki könnyedén módosíthatja a saját böngészőjében illetve esetleg ezen információk lehetnek elavultak is.

A többnyelvűség

A többnyelvűséget támogatandó a program a Smarty-féle megoldást használja, azaz egy nyelvi file-ban vannak deklarálva a karakterláncok `[hu]`, `[en]` . . . fejlécekkel elválasztva. Amit a `header.tpl` sablon file betöltésekor a például `{config_load file=lang.conf section=hu}` résszel tölt be. Ezt egy kicsit tovább bonyolítottam azzal, hogy először betölti az angolt, majd a választott nyelvet, így az esetlegesen még le nem fordított szövegek angolul jelennek meg. Továbbá egy `lang` osztály is feldolgozza a nyelvi file-t és a `getString(string)` függvény segítségével a `php`-n belül is elérhetővé teszi a fordításokat. Ezen utóbbira azért van szükség, hogy a `php`-n belűl előállított Flash-es grafikonokon belül is le legyen fordítva a szöveg. Egy példa látható arra, hogy mi történik, ha hiányos a fordítás a 2.2 ábrán.



2.2. ábra: Hianyzó fordítás

A geográfiai adatok

A látogatók területi eloszlásának megjelenítéséhez szükség van egy adatbázisra, ami a gépneveket országokhoz rendeli. Ezen adatbázist a http://ip-to-country.webhosting.info/ oldalról szereztem be, ahol ezt ingyenesen a rendelkezésére bocsátják bárkinek. Persze léteznek fizetős adatbázisok is, amik pontosabb adatokat szolgáltatnak város illetve internetszolgáltatói adatokkal, de az én szempontomból elég volt az országok ismerete.

Az adatokat egyrészt az „utolsó találatok” oldalon jelenítem meg, másrészt a „látogatók” oldalon készítek belőle egy sematikus térképet. A térkép megjelenítéséhez a DIY Map (http://www.backspace.com/mapapp/) ingyenes Flash alapú térképet használom. Sajnos a tesztelés alatt jöttem rá, hogy a készítő nem mindig tudta a hivatalos kétbetűs országkódokat és így az adatbázis és a térkép között konvertálnom kell azokat.

5.5. Megjelenítés

A web-oldal teljes egészében a korszerű CSS[7] technológiára épül. Táblázatokat csak a feltétlenül szükséges (tényleges táblázatok) helyeken tartalmaz. A grafikonok megjelenítéséhez korszerű Flash alapú megoldást használtam. Az oldalak megjelenítését a háttérben fejlett sablonrendszer végzi, szem előtt tartva az első fejezetben a korszerű web-fejlesztésről leírtakat.

Mint ahogy leírtam az első részben, a különféle böngészők nemkompatibilitása miatt, főleg az Internet Explorer hibái miatt, sokszor kellett órákig kutatni a trükkös megoldásokat, hogy minden platformon ugyanolyan megjelenítést kapjak.

6. Adatbázis

Az egész program lelke az adatbázis. Egy átlagos nem felkapott weboldal[8] napi 3000-5000 találatot[9] kap, a felkapott web-oldalak akár több százszorosát is. A statisztikát készítő programnak fel kell lennie készülve ezen adatok akár több éves időintervallumon való kezelésére.

Nem lenne kifizetődő file-okat használni adattárolásra, mivel azok elérése sokkal lassabb és körülményesebb, mint az adatbázis használat, nem is beszélve bonyolult „lekérdezések" végrehajtásáról.

Az adatbázis-tervezésnél számba kell venni először is, hogy mik azok az adatok, amiket ki szeretnénk nyerni a lekérdezések segítségével. Esetemben a legfontosabbak a következők:

- oldalankénti látogatószám
- oldalankénti és felhasználónkénti hozzáférés-sorozat, értem ez alatt a látogató böngészője által kezdeményezett kérések idő szerinti rendezett sorozatát
- a meglátogatott URL-ek valami rendezés szerinti sorozatát (látogatásszám, látogatási idő)

A fenti adatokra a másodperc töredéke alatt, esetleg maximum 1-2 másodperc várakozás után, kíváncsi vagyok.

Mivel a különálló oldalak adatai nem fedik át egymást és bár a browser adatok és az IP szám (ez utóbbinak elég kicsi az esélye) átfedhetik egymást, de mégis ezek szétszedése nem okoz túlságosan nagy redundanciát. Továbbá az oldalankénti szeparálás nagyban lecsökkenti a fenti lekérdezések idejét.

Ezen megfontolások szerint a következő adatbázis struktúrát alkalmaztam:

`sites` tábla Ez tárolja a rendszerbe regisztrált weboldalak adatait, ezek a következők: azonosító (`id`), URL (`url`), legyen-e cookie-s felhasználó követés (`cookie`), milyen nyelven jelenjen meg a weboldal (`lang`), a belépéshez szükséges azonosító illetve jelszó (`nick, passwd`) illetve a felhasználó e-mail címe (`email`)

[7] Cascading Style Sheets

[8] Körülbelül 200-300 ember által rendszeresen látogatott közösségi vagy véges oldal.

[9] Találat alatt a web-szerver által kiszolgált kéréseket értem.

`visits_nnnn` **tábla** Tárolja az egyedi látogatások adatait, amik: azonosító (`id`), dátum (`date`), idő (`time`), látogató (`visitor`), látogatás szám (`visitno`), látogatáson belüli oldallekérés (`hitno`), URL (`url`), az URL, ahonnan átkattintottak (`referer`)

`visitors_nnnn` **tábla** Ez a tábla tárolja az egyes látogatók azonosításához szükséges adatokat, úgymint: azonosító (`id`), domain-név (`host`), browser illetve ügynök azonosító (`agent`), a felhasználót azonosító cookie-k száma, ha van (`cookie`)

`hostss_nnnn` **tábla** A látogatók domain-neveit tartalmazó tábla, mezői a következők: azonosító (`id`), domain-név(`host`)

`agents_nnnn` **tábla** A látogató kliense által küldött (`User-agent:`) szövegeket tartalmazó tábla, amely a következő adatokat tartalmazza: azonosító (`id`), az azonosító szöveg (`agent`)

`urls_nnnn` **tábla** Az egyedi URL-eket tartalmazza: azonosító (`id`), URL (`url`)

`ip_to_country` **tábla** A http://ip-to-country.webhosting.info oldalon elérhető ingyenes IP cím - ország hozzárendeléseket tartalmazó táblázat adatait tartalmazza, ami egy IP-cím intervallum (speciálisan kódolva), a két, illetve három betűs országkód illetve a teljes országnév.

A fenti leírásban szereplő táblák közti kapcsolatot a következő idegenkulcsot szolgáltatják:

- `visits_nnnn:visitor` – `visitors_nnnn:id`
- `visits_nnnn:url` – `urls_nnnn:id`
- `visitors_nnnn:host` – `hosts_nnnn:id`
- `visitors_nnnn:agent` – `agents_nnnn:id`

A táblák nevében látható `_nnnn` a `sites` táblában lévő `id`-re utal, így vannak szétválasztva egymástól a weboldalak adatai.

A `visits_nnnn` táblában szereplő látogatásszám mező azt mutatja, hogy hányadik látogatása ez az adott látogatónak. Kezdetben ez 1-ről indul és minden általa generált találatkor az előző értéke íródik ide, amennyiben még nem telt el egy meghatározott idő, illetve 1-el nagyobb szám, ha már eltelt. Ez az idő határozza meg, hogy mennyi inaktivitásnál tekintjük már a találatot egy új látogatás részének.

6.1. Fontosabb lekérdezések

Itt a trükkösebb, de a kimutatások szempontjából fontosnak tartott lekérdezéseket fogom bemutatni. A lekérdezések SQL szintaxis szerint kerülnek bemutatásra, kisebb magyarázószöveg kíséretében. A lekérdezések mindig egy-egy konkrét számadatokkal ellátott

példát mutatnak be. Mint majd látható lesz, a lekérdezések néha elég bonyolultak lettek, ellenben nagyon gyorsan hajtódnak végre, ez az adatbázisstruktúrának köszönhető. Illetve az is bonyolítja egy kicsit a lekérdezéseket, hogy egyes számolásokat az adatbázis-kezelővel hajtatok végre, ilyenek például az adatok százalékosítása, és egyek karakterláncokon elvégzett, a kiiratás szépítő műveletek.

Az alant látható lekérdezéseket máshogyan is meg lehet valósítani. Van, ahol nem is próbáltam más megoldást találni, van, ahol a különféle megoldások összevetése után választottam ki a leggyorsabbat.

Az első lekérdezés a megadott időpontok közötti új illetve visszatérő látogatók százalékos arányát szolgáltatja.

```
 1 select
 2   truncate(100 * t1.new / t1.all,1) as 'New',
 3   truncate(100 * t1.returning / t1.all,1) as 'Returning' from (
 4     select
 5       count(*) as 'all',
 6       count(visitno = 0 or NULL) as new,
 7       count(visitno > 0 or NULL) as returning from (
 8         select * from stat_visits_2 where
 9           date >= "2008-02-22" and date <= "2008-03-06"
10            group by visitor, visitno
11          ) as e
12     ) as t1;
```

A fenti lekérdezés egy háromszorosan egymásba ágyazott lekérdezés. A legbelső rész **8. sor** adja vissza a megadott két dátum közötti látogatókat azonosító és látogatási szám szerint csoportosítva. Mivel ilyenkor a visszaadott sor a legelső lesz, így ez a táblázat csak a belépési oldalakat tartalmazza. Az eggyel kijjebb lévő select (**4. sor**) visszaadja az összes sor számát, valamint az olyanok számát, ahol a `visitno` 0 illetve nagyobb, mint 0. Ezek után a legkülső select már csak százalékosítja az eredményeket és ellátja a megfelelő címkével is, mivel a tényleges megjelenítésnél az itt szereplő oszlopcímkék kerülnek mutatásra.

Ez a többszöri egymásba ágyazás az esetleges másként nem megvalósíthatóságon túl azért is van, hogy a külső lekérdezések minél inkább szűkített adathalmazon dolgozzanak, ezzel gyorsítva a lekérdezést.

A következő lekérdezés a böngészők százalékos eloszlását adja vissza.

```
1 select
2   truncate(100 * tt1.firefox / tt1.all,1) as 'Firefox',
3   truncate(100 * tt1.opera / tt1.all,1) as 'Opera',
4   truncate(100 * tt1.konqueror / tt1.all,1) as 'Konqueror',
5   truncate(100 * tt1.explorer / tt1.all,1) as 'Explorer',
6   truncate(100 * tt1.epiphany / tt1.all,1) as 'Epiphany',
7   truncate(100 * (tt1.all - tt1.firefox - tt1.opera - tt1.konqueror -
```

```
 8                  tt1.explorer - tt1.epiphany) /
 9                  tt1.all,1) as 'Other' from (
10     select
11       count(*) as 'all',
12       count(t3.agent like '%Firefox%' or NULL) as firefox,
13       count(t3.agent like '%Opera%' or NULL) as opera,
14       count(t3.agent like '%Konqueror%' or NULL) as konqueror,
15       count(t3.agent like '%MSIE%' or NULL) as explorer,
16       count(t3.agent like '%Epiphany%' or NULL) as epiphany from
17         stat_visits_2 where
18           date >= "2008-02-22" and date <= "2008-03-06" as t1,
19         stat_visitors_2 as t2,
10         stat_agents_2 as t3 where
21           t1.visitor = t2.id and t2.agent = t3.id
22     ) as tt1;
```

Itt a legfőbb eltérés a legutóbbi lekérdezéshez képest az, hogy a tényleges böngészőadatok egy karakterláncban helyezkednek el, ami körülbelül a következő formájú: **Mozilla/5.0 (X11; U; Linux i686; hu; rv:1.8.1.12) Gecko/20080207 Ubuntu/7.10 (gutsy) Firefox/2.0.0.12**. Ezért itt, mint a **12.-16. sorig** látszik a MySQL `like` funkcióját kell alkalmazni. Továbbá a találati számot a `stat_visitors_2` táblából tudjuk meg, amihez a böngészőadatok 3-as láncolással kapcsolhatók (**17.-21. sor**).

A következő lekérdezés az órákra bontott látogatószámot kéri le, itt már az egyszerűbb olvashatóság miatt elhagytam a dátum szerinti szűkítést, de a fenti két példa alapján bárki egyszerűen beleteheti azt a lekérdezésbe.

```
1 select o.ora, count(*) from (
2   select substring(time,1,2) as ora from
3     stat_visits_4 group by visitor, visitno
4   ) as o group by o.ora;
```

Itt egy az eddigiekben be nem mutatott SQL függvény alkalmazok, ez a `substring`, aminek a segítségével veszem ki az ÓÓ-PP-MM formátumú időbélyegből az óra részt. A látogatók első találatára való leszűkítést itt is a `group by visitor, visitno` rész végzi.

A keresőkifejezések ranglistája:

```
1  select keywords, count(*) as db from (
2    (select v.id,
3     substring(substring(r.referer,instr(r.referer,'q=')+2),1,
3               instr(substring(r.referer,instr(r.referer,'q=')+2),'&')-1)
4     as keywords from
5       stat_referers_4 as r, stat_visits_4 as v where
6         v.referer = r.id and
```

```
7           r.referer like '%q=%' and r.referer like '%google.%'
8     ) union (
9 ...
10    ) as t where keywords not like '' group by keywords order by db
```

A fenti lekérdezést erős php segítséggel állítom elő. A `Helper` osztályban van egy tömb, ami a keresőmotorokat tartalmazza a következő formában: minden tömb elem maga is egy 2 hosszú tömb, aminek az első eleme a kereső azonosításához szükséges karaktersorozatot tartalmazza, úgy mint "google.", "tango.hu", ..., a második eleme a hivatkozás karakterláncban a keresőszavak változóját tartalmazza, és például a google esetében "q=", mint a fenti lekérdezésben is látszik.

A lekérdezésből a közepén **9. sor** kivágtam egy darabot, ott a fenti **2.-7. sorhoz** hasonló részek ismétlődnek, ezeket egy ciklussal állítom elő az előbb említett tömb segítségével. Maga a lekérdezés nem túl izgalmas, csak string manipulációk vannak benne, de ami szükségessé tette, hogy itt leírjam az az, hogy az eddigiekben nem látott `union` kulcsszót használok benne. Ezzel azt érem el, hogy a benti lekérdezések eredményei, amik keresőspecifikusan adják vissza a kulcsszavakat, összevonódnak. Így a lekérdezés eredményeképpen egyben tudom megkapni a keresőkifejezések ranglistáját.

7. Az intelligens statisztika mód

Kiemelt célom volt, hogy a program ne csak a számadatokat, táblázatokat és grafikonokat mutasson meg, mivel nem mindenki képes azokból a szükséges következtetéseket levonni, hanem a következtetéseket is vonja le az adatokból. Így arra jutottam, hogy a következő fontosabb adatokat tekintem az ún. intelligens kimutatásnak:

- új hivatkozások,
- megújuló hivatkozások,
- eltűnt (inaktív) hivatkozások,
- az aktív hivatkozások száma valamint a legtöbb találatot adó 5,
- trend jelzés,
- utolsó 7 nap látogatószáma, napi átlag illetve átlagos látogatáshossz.

Ezek között kiemelt fontosságúnak tekintem az új hivatkozásosokat és az eltűnt hivatkozásokat, mivel ezeket általában más programok nem jelenítik meg, azokat csak akkor lehet megállapítani, ha az ember heti szűkítésben nézi és hasonlítgatja össze a hivatkozók táblázatát. Pedig véleményem szerint ezek eléggé fontos és árulkodó adatok lehetnek a web-oldal üzemeltetőknek.

7.1. A hivatkozásváltozások

A hivatkozásváltozáshoz a program az adatokat a következő módon szolgáltatja:

- Az aktív hivatkozások száma, azon domain nevek száma, ahonnan az elmúlt 7 napban látogató érkezett az oldalra.
- Az új hivatkozások listája azon domain neveket tartalmazza, ahonnan az elmúlt 7 napban jött látogató, de előtte még sohasem.
- A megújuló hivatkozások az újhoz hasonlóan azt nézi, hogy honnan jött hivatkozás az elmúlt 7 napban, de itt azokat vesszük csak be, ahonnan az elmúlt 7 napban nem jött, de előtte már igen.
- Az eltűnt hivatkozások azok, akiktől az elmúlt héten nem jött látogató, de az előtte lévő héten igen.

Persze lehetne más időintervallumot is választani, de ennél több már túl hosszú lenne, a kevesebb meg túl nagy varianciát produkálna. Valamint így összhangban tud lenni az itt mutatott többi statisztika időintervallumaival.

7.2. Trend kimutatás

A trendek megállapítására a tőzsdei világban a mozgó-átlagokat használják leginkább. A legalapvetőbb indikátor arra, hogy a trend milyen irányú, két különböző nagyságú mozgó-átlag egymáshoz viszonyított helyzete. Esetünkben az egyik a 7 napos (továbbiakban $MA\,(7)$), itt fontos, hogy ne válasszunk túl kicsit, mert a héten belüli internetezési szokások hamis trendet mutathatnak[10], valamint, hogy ne válasszunk túl nagyot, mivel akkor nagy lesz a késés. A másik a 28 napos mozgó-átlag (továbbiakban $MA\,(28)$).

Amennyiben

$$MA\,(28) > MA\,(7)$$

csökkenő a trend, illetve,

$$MA\,(28) < MA\,(7)$$

esetén növekvő. Ha a két oldal egyenlő, akkor épp változás figyelhető meg a trend irányában.

Az, hogy az a módszer pár nap késésben van a tényleges trendekhez képest, elfogadható a web-statisztikában, ugyanis az emberek általában arra kíváncsiak, hogy például az elmúlt héten hogyan alakult a látogatottság, ott egy *„az elmúlt héten növekedés volt megfigyelhető”* forma válasz teljesen kielégítő.

A program a hosszútávú trendet a fent bemutatott módon szolgáltatja, a rövidtávúra a következő eljárást használja. Amennyiben a következő egyenlőtlenség áll fenn, akkor az növekedést jelent, ellenkező esetben csökkenést:

$$|MA7(\text{tegnap}) - MA28(\text{tegnap})| > |MA7(\text{tegnapelőtt}) - MA28(\text{tegnapelőtt})|$$

[10]Például, ha 3 napos mozgó-átlagot veszünk és a web-oldalunkat főként hétvégén látogathatják, akkor héten belül is nagy ingásokat fogunk kapni, holott minket a hosszútávú trend érdekel leginkább.

illetve, ha egyenlő a két oldal (amire elég kicsi az esély, de mégis számolni kell vele), akkor növekvést jelent, ha

$$MA7(\text{tegnap}) > MA6(\text{tegnapelőtt}).$$

8. A megvalósított felhasználómodellezés

A 7.7.. fejezetben bemutatott PrefixSpan algoritmust valósítottam meg az általam írt programban. Azért esett erre a választásom, mert középúton áll az egyszerű megvalósíthatóság és a jó teljesítmény között. Az egyszerű megvalósításon azt kell érteni, hogy az általam választott adatbázis-struktúra esetén egyszerű MySQL lekérdezések segítségével előállítható a kívánt végeredmény. Azért volt ez fontos, mert az adatbázisra támaszkodva sokkal egyszerűbb lesz a program és gyorsabb is, mivel az adatbázismotorok pont arra vannak optimalizálva, hogy a kéréseket szolgálják ki, míg ha bonyolult php kódokat használtam volna, akkor annak az interpretált jellege miatt nagyon sokat vesztettem volna a teljesítményből. A mostani megvalósítás csak az SQL lekérdezéseket állítja elő programból, valamint a végső megjelenítéshez gyűjti össze az adatokat.

Az eredeti algoritmust kiegészítettem egy újdonsággal, ami a végső adatok megjelenítését javítja, ez a minimális minta hossz, ezzel kerülve el a sok rövid találat megjelenítését.

A továbbiakban a könnyebb olvashatóság érdekében, a kódrészletekben `ID`, `MINSUP`, `MINHOSSZ` és `MINDATUM` karaktersorozat írok, de a tényleges megvalósításban oda a php program illeszti be a tényleges értékeket.

Első lépéként az általam megadott százalékos *minsup* értéket tényleges számmá (`MINSUP`) alakítom, a

```
select count(*) as db from (select * from stat_visits_ID where
date > MINDATUM group by visitor,visitno) as n;
```

lekérdezés által visszadott összes sorozatszám segítségével. Ezután meghívom a

```
prefixspan(PREFIX, VETITETTADATBAZIS, MINTAK)
```

függvényt, ahol a `PREFIX` az éppen vizsgált prefix, a `VETITETTADATBAZIS` az aktuális vetített adatbázistábla neve (mivel a vetített adatbázisokat külön táblában tárolom az algoritmus futása idejére) és a `MINTAK` a tömb, amiben a megtalált mintákat tárolom és a programon belül hivatkozásként adom át azon függvényeknek, aminek szüksége van rá.

A `prefixspan` függvény a első lépésként a

```
gyakorielem(TABLANEV)
```

függvény segítségével, kiszámolja a lokálisan gyakori elemeket, azaz megkapja a következő SQL lekérdezés eredményét:

```
select url,sup from (select url, count(*) as sup from TABLANEV where
date > MINDATUM group by url order by sup) as t where sup >= MINSUP);.
```

Majd végigmegy a gyakori elemeken azokat konkatenálja PREFIX-el, beleteszi a MINTAK tömbbe, majd meghívja rekurzíve saját magát a KONKATENALTPREFIX, makevetitettadatbazis(KONKTENALTPREFIX, FORRASTABLA) és MINTAK paraméterekkel.

A fent említett

```
makevetitettadatbazis(PREFIX, FORRASTABLA)
```

függvény létrehozza először az átmeneti táblát, aminek a tmp_e1_e1_..._en nevet adja, ahol e1 ...en a PREFIX-ben lévő elemek. Ezen táblának a formátuma (id int, visitor int, visitno int, url int, date date). Majd az

```
insert into atmenetitablanev QUERY;
```

SQL lekérdezést hajtja végre, ahol a QUERY részt a

```
vetitettadatbazisquery(PREFIX, FORRASTABLA)
```

függvény állítja elő a következő módon. A visszaadott lekérdezés a következő lesz:

```
select isa.id, isa.visitor, isa.visitno, isa.url,isa.date from
FORRASTABLA as isa, (PREFIXFV(PREFIX, PREFIXHOSSZ, FORRASTABLA)) as p
where isa.id > p.id and p.visitor = isa.visitor and p.visitno = isa.visitno,
```

ahol a

```
PREFIXFV(PREFIX,PREFIXHOSSZ, FORRASTABLA)
```

részt egy külön függvény állítja elő úgy, hogy, ha PREFIXHOSSZ = 1, akkor a

```
select c.id, c.visitor, c.visitno from FORRASTABLA where
date >= MINDATE as c where c.url = PREFIXELSOELEME;
```

és ha a PREFIXHOSSZ nagyobb mint 1, akkor a következő rekurzióval állítja elő azt

```
select ln.id, ln.visitor, ln.visitno from
  (select id, visitor, visitno from FORRASTABLA where
    date >= MINDATE and url = prefixelsoeleme
  ) as ln,
  (PREFIXFV(PREFIX, PREFIXHOSSZ-1, FORRASTABLA)) as rn
  where rn.visitor = ln.visitor and rn.visitno = ln.visitno and
  rn.id > ln.id;.
```

Álljon itt egy példa a fenti algoritmus által generált MySQL lekérdezésre ami a 26, 2 prefix sorozat vetített adatbázisát generálja.

```
 1 insert into tmp_26_2
 2   select isa.id, isa.visitor, isa.visitno, isa.url, isa.date from
 3     tmp_26 as isa,
 4     (select l2.id, l2.visitor, l2.visitno from
 5       (select id, visitor, visitno from
 6         tmp_26 where ‘date‘ > 2008-03-21 and
 7                      url = 2
 8       ) as l2,
 9       (select c.id, c.visitor, c.visitno from
10         tmp_26 as c where ‘date‘ > 2008-03-21 and
11                           c.url = 26
12       ) as r2 where r2.visitor = l2.visitor and
13                     r2.visitno = l2.visitno and
14                     r2.id > l2.id
15     ) as p where isa.id > p.id and
16                  p.visitor = isa.visitor and
17                  p.visitno = isa.visitno
```

Látható, hogy a lényeg ugyanaz, amit az algoritmus vetített adatbázis gyártás leírásánál is bemutattam, azaz azon sorokat bele `tmp_26` táblából `tmp_26_2` táblába, amikre igaz az, hogy tartalmazza részsorozatként 26-ot és 2-t ilyen sorrendben.

A fenti SQL lekérdezésekkel definiált algoritmus a PrefixSpan pontos implementációja.

8.1. Az algoritmus beillesztése a web-oldalba

A fenti PrefixSpan algoritmus futásideje nagy, akár több percet is igénybe vehet. Itt szembesültem annak a problémájával, hogy hogyan is lehet ezt egy web-oldalba beilleszteni, ahol ha az oldal nem válaszol egykét másodperc alatt, akkor a felhasználók már türelmetlenek. Erre az offline alkalmazások progressbar-t használnának, de hogyan valósítsam én ezt meg web-es felületen?

A fő probléma az, hogy egy szerveroldali alkalmazás állapotát szerettem volna a kliens oldalon megjeleníteni, sőt, olyan alkalmazás állapotát, amit egy függvény meghívásával indítok és egyszerre fut le, nem több kis részből áll. A megoldás, amit találtam AJAX-os elemeket tartalmaz.

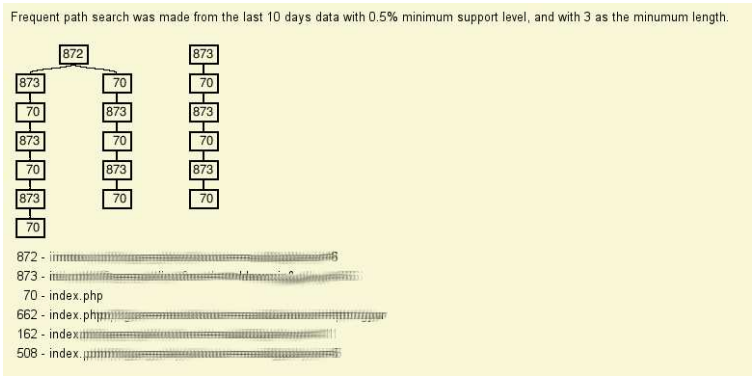
A web-lapba, amire az eredményt várom egy IFRAME-et illesztettem, aminek a forrásában megadott oldal számolja ki a PrefixSpan algoritmus eredményét. Mivel az algoritmus az oszd meg és uralkodj elvén alapul és faszerűen szétbontja a problémát, az első szinten az egyes ágakba való belépésnél ki tudtam számolni a teljes futás százalékos állapotát. Ezt egy `Progressbar` nevű osztály által megvalósított progressbar-on jelenítem meg úgy, hogy CSS segítségével először kirajzolom a teljes csíkot, majd a százalékos értékeknek megfelelően DIV-eket pakolok bele, ahogy megy előre a program, ez a 2.3. árán látszik. A php programban a kimenetei tárolót minden írás után erőszakosan kiüríttetem, így annak

az eredményét a felhasználónak juttatom, ahol a browser azt megjeleníti. Ezzel elértem a szerveroldal által vezérelt progressbar megjelenítését.

2.3. ábra: PrefixSpan futás közben

Itt meg kell jegyeznem, hogy a megvalósítást az esetleges szerver és kliens közé ékelt cache-elést végző proxy-k megzavarhatják és így ebben az esetben a kliens nem látja a progressbar-t, csak majd a végleges oldalt. De ezzel sajnos nem lehet mit kezdeni. Lehetne egy másfajta megvalósítást alkalmazni, ami a szerver oldalra egy külső file-ba vagy adatbázis táblába mentené a függvény állapotát, és azt AJAX-os hívásokkal kérdezném le és frissíteném a progressbar-t, de ez kevésbé lenne megbízható és sokkal több erőforrást is fogyasztana a PrefixSpan számolását végző, egyébként is leterhelt szerveren. Valamint ezen esetben az AJAX-is hívások aszinkron mivolta miatt a kijelző állapota különbözne a tényleges állapottól.

Miután az algoritmus lefut és a progressbar feleslegessé válik, azt egy kis JavaScript programmal tüntetem el, és jelenítem meg helyette az eredményt, az eredménye a 2.4. árán látszik.



2.4. ábra: PrefixSpan által készített gráf

Ezt a megoldás még máshol nem láttam az interneten, tudomásom szerint én csináltam először ilyet. A leghasonlóbb megoldás, amit találtam, az a MediaWiki installálásánál a böngészőnek küldött folyamatos szöveges helyzetjelentés.

Sajnos a fejlesztéshez használt szerver egy virtuális gépen helyezkedik el, és emiatt eléggé korlátosak az erőforrásai, a maximális php futási idő és korlátozott memóriahasználat, emiatt a generálás elég gyakran megakad, mivel a szerver lelövi a szkriptet, illetve végtelennek tűnő lekérdezésbe fut bele a MySQL. Sajnos ennek elkerülésére szoftveresen nem

lehetett mit tenni, bár fejletteb algoritmus implementálása gyorsíthatná némileg, de végülis csak a hardware erősítése segítene.

Utószó

Az dolgozatomban bemutattam, előszóban vázolt piaci okok miatt a mai internetes világban elengedhetetlen statisztika készítés készítés módjait és technikai hátterét. Ismertettem a megoldandó problémát és ezzel együtt az elterjedt megoldásokat is, azok hátrányait és előnyeit is. Egy tényleges megvalósítás keretein belül kitértem a korszerű web-es tartalommegjelenítés főbb aspektusaira, úgy mint: AJAX-os technikák, CSS-es tartalomformázás, sablonos oldalfelépítés, adatbázis háttér. Az adott technikáknál röviden ismertettem az adott témakör történetét és az alkalmazása mellet illetve ellene szóló érveket, valamint azt, hogy az hogyan befolyásolta az internetes tartalomfejlődést. Részletesen kitértem a felhasználói viselkedéselemzésnél használt algoritmusokra, bemutatva azok működését és matematikai hátterét, valamint példával illusztráltam őket.

Külön fejezetet áldoztam a szekvenciális mintabányászat problémakörének, amit a többek között a felhasználói viselkedésmodellezésnél is használnak. Bemutattam magát a problémát és a megoldásra adott legfontosabb, mérföldkőnek számító algoritmusokat. Mindezt időrendi sorrendben tettem. Az algoritmusokat példákkal illusztráltam és mindvégig szem előtt tartottam, hogy minél inkább megőrizzem az eredeti irodalmakban lévő jelölésrendszert és példákat, hogy a jobban elmélyülni vágyó olvasók első lépéseit megkönnyítsem.

Majd az utolsó fejezetben a saját web-statisztika készítő online alkalmazásom legfőbb részleteit mutattam be. Az adatbázishátteret, az általam a gyakori felhasználói utak kinyerésére megvalósított PrefixSpan algoritmust, valamint a program egy-két másik érdekességét, úgy mint az adatgyűjtési eljárásokat, a cookie-k. Nem tértem ki mindenre, de az egyes fejlesztés közbeni implementációra vonatkozó döntéseimet indokoltam és bemutattam az esetleges egyéb megoldási lehetőségeket is.

Az általam kitűzött célt megvalósítottam, azaz egy komplett web-statisztika készítő alkalmazást készítettem úgy, hogy előtte belemélyedtem a mögöttes problémakörbe. Az alkalmazás készítésénél szem előtt tartottam, hogy az minden modern követelménynek megfeleljen. Alkalmaztam a mai divatos és fejlett technikákat, mint az AJAX, CSS, objektumelvű programozás és a sablonos web-oldal kialakítás. A web-oldal támogatja a többnyelvűséget, ide értve a távol keleti nyelveket is, a külalakja a mai elterjedt letisztult formavilágot idézi, szolíd Flash-es grafikonokkal tarkítva. A leglényegesebb, hogy tartalmazza a még kevés másik ilyen szolgáltatásban megtalálható felhasználói modellezést és az egyedülálló intelligens statisztikát.

Irodalomjegyzék

[1] Jian Pei, Jiawei Han, Behzad Mortazavi-asl and Hua Zhu: *Mining Access Patterns Efficently from Web Logs*, Proceedings PacificüAsia Conference on Knowledge Discovery and Data Mining (PAKDD'00), 2000.

[2] Federico Michele Facca, Pier Luca Lanzi: *Mining interesting knowledge from weblogs: a survey*, 2004.

[3] R. Agrawal and R. Srikant: *Mining sequential patterns.*, ICDE'95, pp. 3-14, Taipei, Taiwan, 1995. márcisu.

[4] Xiangji Huang, Aijun An, Nick Cercone: *Comparison of Interestingness Functions for Learning Web Usage Patterns*, 2002.

[5] Rakesh Agrawal, Ramakrishnan Srikant: *Fast Algorithms for Mining Association Rules*, Proceedings of the 20th International Conference on Very Large Databases, Santiago, Chile, Sept. 1994.

[6] Jian Pei, Jiawei Han, Behzad Mortazavi-Asi, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, Mei-Chun Hsu: *Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach*, 2003.

[7] R. Agrawal and R. Srikant: *Mining sequential patterns: Generalizations and Performance Improvements*, 1996.

[8] Jilanyong Wang and Jiawei Han: *BIDE: Efficent Mining of Frequent Closed Sequences*, 2004.

[9] Brian Clifton: *Web Traffic Data Sources & Vendor Comparison*, 2007 november 13.

[10] http://en.wikipedia.org/wiki/AJAX (2008.04.03)

[11] http://en.wikipedia.org/wiki/Cascading_Style_Sheets (2008.04.03)

[12] http://en.wikipedia.org/wiki/Web_analytics (2008.04.03)

[13] Petre Tzvetkov, Xifeng Yan, Jiawei Han: *TSP: Mining Top-K Closed Sequential Patterns*, 2005.

[14] Jinlin Chen, Terry Cook: *Mining Contignous Sequential Patterns from Web Logs*, 2007.

[15] Shengnan Cong, Jiawei Han, David Padua: *Parallel Mining Of Closed Sequential Patterns*, 2005.