A RABIN-KARP MINTAKERESŐ ALGORITMUS HASZNÁLATA SZÖVEGELEMZŐ SZOFTVEREK KONTEXTUSÁBAN

Szerző:

Mező Péter Dániel
Forward University

Első szerző e-mail címe:
peter.mezo1@gmail.com

Lektorok:

Pšenáková Ildikó (Ph.D.)
Trnava University (Szlovákia)

Pšenák Péter (Ph.D.)
Comenius University Bratislava (Szlovákia)

és további két anonim lektor…

Absztrakt

Egy szövegben rejlő mintázat (például egy szövegrészlet ismételt, többszöri előfordulása) megkeresésére a Rabin-Karp algoritmus a hashing módszert használja. A hashing módszer egy rendszerint szöveges adatkészletet más formátumba konvertál. Noha ennek a módszernek mély gyökerei vannak a kiberbiztonságban, emellett a szöveges mintakereső algoritmusok létrehozására is felhasználhatók.

Kulcsszavak: szövegelemzés, hashing, Rabin-Karp

Diszciplína: informatika

Abstract

UTILIZATION OF THE RABIN-KARP PATTERN FINDING ALGORITHM IN CONTEXT OF TEXT ANALYZING SOFTWARES

The Rabin-Karp algorithm utilizes the method of hashing in order to discover patterns (for example a reoccurring text section) within a text document. The method of hashing converts a set of (usually) text-type data into another format. Although it has deep roots in cybersecurity, it can be used to create textual pattern-finding algorithms too.

Keywords: text analysis, hashing, Rabin-Karp

Discipline: computer science

Mező Péter Dániel (2023): A Rabin-Karp mintakereső algoritmus használata szövegelemző szoftverek kontextusában. *OxIPO – interdiszciplináris tudományos folyóirat, 2023/2.* 73-78. DOI 10.35405/OXIPO.2023.2.73

E tanulmány a Rabin-Karp típusú mintakereső algoritmusok működését vizsgálja a szövegelemző szoftverek kontextusában. A Rabin-Karp algoritmus a „hashing"-nek nevezett módszeren alapszik, amely a „string" (szöveges) típusú adatok más formátumba változtatását teszi lehetővé. A következőkben a hashing algoritmus működése, a Rabin-Karp típusú algoritmus sajátosságai és valamint a Rabin-Karp algoritmus előnyei és hátrányai lesznek bemutatva.

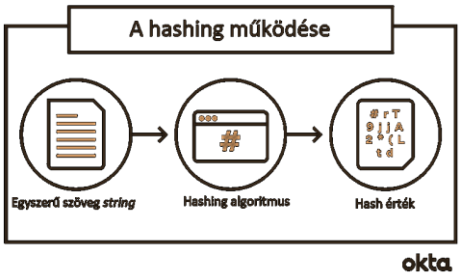
Hashing algoritmusok

A „hashing" algoritmusok alapja a szöveges (string) típusú adatok átformázásán alapszik (Zola, 2021). A kibervédelem és kriptográfia területeken gyakran használt ez a metódus egy adott adatsorozat matematikai műveletek segítségével történő titkosításához (Zola, 2021; 1. ábra).

A „hash" algoritmusok magukban egy irányúak, avagy a beviteli szöveget (a „message"-t) csak lefordítják egy hash érték-re (vagy egyszerűen: egy hash-re); a hash visszafordítása a message-re a „kulcs" nélkül vak találgatás. A konverzió során egy relative tetszőleges hosszúságú message egy előre megadott hosszúságú hash-re történő átalakítása valósul meg (Loo, 2023).

Ez biztosítja az adatok biztonságát is – erre jó példa a SHA-1 hash algoritmus, amely kötött 160-bit rendszere 40 karakterre konvertáljani a beviteli értéket (Loo, 2023).

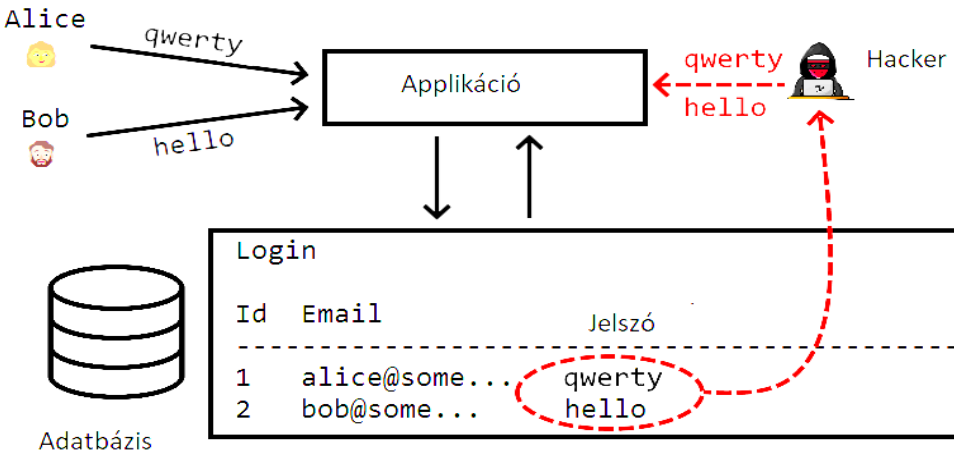
1. ábra: A hashing alapvető működése. Forrás: Net1 (fordította: a Szerző)



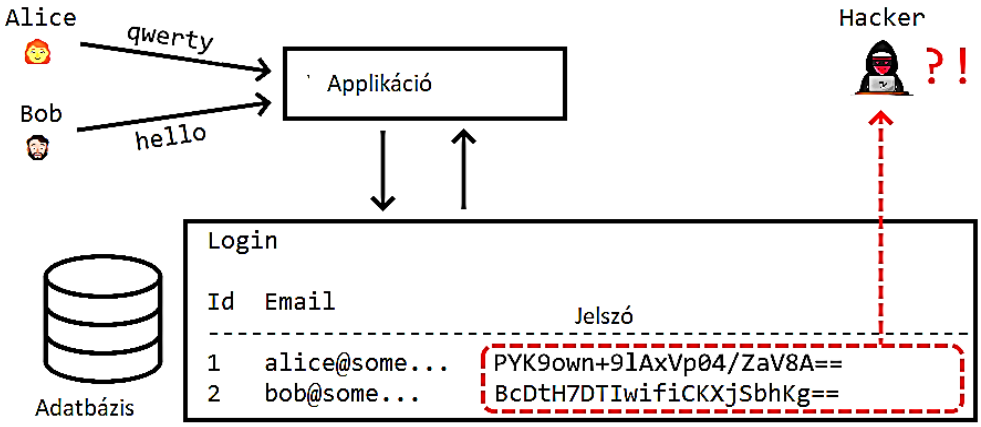
Konkrét példa a hashing bemutatására például egy alapvető jelszó-értékelő rendszer. Tételezzük fel, hogy egy vállalat a saját szerverén tárolja a felhasználói jelszavát és a felhasználók minden egyes belépésnél kötelezőek beírni a jelszavukat. Ha a vállalat a jelszavakat titkosítás nélkül tárolja a szerveren, hogy minden felhasználói belépésnél azokkal vesse össze a beírt jelszó próbákat, egy esetleges kibertámadás alkalmával minden felhasználói jelszót veszélyeztetnek (Net2; lásd: 2. ábra).

A hashing módszer használatával a vállalati szerveren a felhasználók jelszavának hash értékét tárolják. A hash a titkosítási kulcs nélkül a kibertámadóknak értelmetlen, így a feltörés veszélyét elhárítja, a kibertámadás megelőzését és az adatvédelmet fokozza (3. ábra). A felhasználó azonosításhoz pedig az algoritmusnak nem kell visszafordítania a szerveren tárolt hash-t az eredeti formájába, hanem a felhasználó által megadott jelszót fordítja le a hash értékére, amit összevet a szerveren tárolt értékkel.

2. ábra: A vállalati jelszavak (például: „qwerty” vagy „hello”) titkosítás nélkül viszonylag könnyen feltörhetők. Forrás: Net2 (Fordította: a Szerző)



3. ábra: A vállalati jelszavak (például: „qwerty” vagy „hello”) titkosítással védettebbek a kibertámadások ellen. Forrás: Net2 (Fordította: a Szer-ző)



A hashing módszer bemeneti értéke végtelen variációjú és hosszúságú lehet, de a hash, amit az algoritmus képez, csak megszámlálható mennyiségű formát ölthet és kötött hosszúságú lehet (Burnett and Foster, 2004). Ez a „kollízió” nevű problémát veti fel: két megegyező hash-t tulajdoníthat két különböző bemeneti értéknek (Loo, 2023). Ez a program céljától függően teljesen megbízhatatlanná teheti aannak

működését. Erre az algoritmus természetétől függően egészen kevés esély történhet, és ki lehet ezt védeni különböző, a tervező által beépített mechanizmusokkal.

A következő fejezetben ennek az ellentéte lesz megvizsgálva, a Rabin-Karp hashing alapú algoritmus, amely a kollíziók jelenlétét használja fel effektív mintakeresésre.

A Naive String-Matching és a Rabin-Karp algoritmus

Ebben a fejezetben a hashing algoritmus szövegen belüli mintakereső aspektusa nagyobb hangsúlyt fog hapni, mint a kibervédelemben betöltött szerepe.

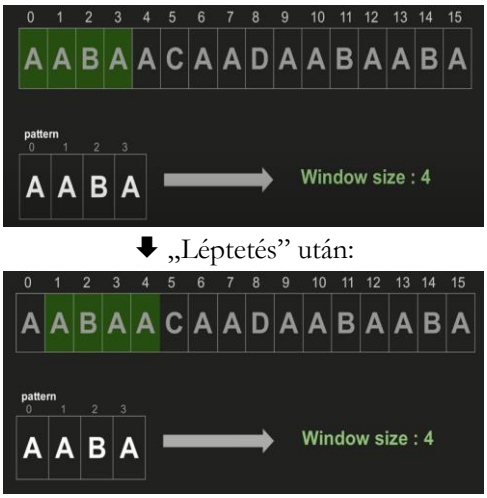
A szövegen belüli mintakeresés legegyszerűbb, de nem legeffektívebb módja az úgynevezett „Brute force" (magyarul: a „nyers erő") algoritmusok használata. A szöveges mintakereső brute force algoritmusok közül gyakran használt a „naive string-matching" módszer (Net3). Ez az algoritmus például a következőképp működik:

1) Egy, a keresett minta hosszával megegyező „ablakot" léptet előre a bemeneti szövegen.

2) Összeveti a bemeneti szöveg első karakterét a keresett minta első karakterével, s ha azok megegyeznek, utána a másodikat a másodikkal, egészen amíg az „ablakon" végig nem ér. Ha minden karakter egyezik, találatot jelez és előlépteti az „ablakot" úgy, hogy a következő permutációban az első vizsgált elem a bemeneti szöveg második karaktere legyen (4. ábra).

3) Ha az egyik karakter nem egyezik, előlépteti az „ablakot" úgy, hogy a következő permutációban az első vizsgált elem a bemeneti szöveg második karaktere legyen.

4. ábra: A keresett mintaablak (négy zöld téglalap) és léptetése a 'naive string-matching' algoritmusban. Forrás: Net4



A Big-O jelölés a programidő komplexitását hivatott bemutatni, avagy: azt, hogy a bemeneti érték változásával hogyan változik a program effektivitása (Simplilearn, 2023). Az O(1) konstans időkomplexitást jelez, avagy a bemeneti érték nincs hatással a program lefutási idejére. Ennek a „naive string-matching" algoritmusnak a maximális hatékonysága a Big-O notáció haszná-

latával O(N), ahol az N a beviteli szöveg mérete. Ezt akkor éri el a program, amikor az input szöveg első karakterével egyik karaktere sem egyezik a szöveggel. A legrosszabb esetben pedig O(M*(N-M+1); amikor is a bemeneti szöveg minden karaktere egyezik a minta minden karakterével (Net4).

Ez az algoritmus még nem implementált hashing-et. A Rabin-Karp ezzel –ellentétben egyszerű karakter összevetés helyett először egyezést keres a minta hash értéke és a szövegablak hash-e között. Míg a kibervédelem területén a kollízió jelenléte a szoftver potenciális teljes használhatatlanságát jelenti, a Rabin-Karp algoritmus ezek jelenlétét használja ki az effektív keresés véghezviteléhez. Ha a két érték megegyezik, csak akkor kezd el karakterenként is egyezést keresni. Ha nincs, előrelépteti az ablakot. Ezzel a trükkel a Rabin-Karp algoritmus megakadályozza, hogy egy ablak-iterációban a program egyesével ellenőrizze a karakter egyezést feltételezve az esélyt, hogy például az utolsó az egyetlen, ami nem egyezik. Mindazonáltal a Rabin-Karp algoritmus az úgynevezett „hamis találat" hibába ütközhet bele (Net5). Van arra esély, hogy a minta hash értéke megegyezik a szöveg-ablak hash értékével, pedig a két string között egyezés nincs. A Rabin-Karp módszer Big-O értéke a legjobb esetben és átlagosan O(M+N), míg a legrosszabb eshetőség az O(N*M); az M a minta hossza, az N pedig a bemeneti szöveg hossza (Net5).

A Rabin-Karp algoritmus előnye

A fentiekben vizsgált két algoritmus (a naiv estring-matching és a Rabin-Karp algoritmus) két eszköz eltérő feladatokra. A fenti példák alapul vették, hogy a bemeneti szövegben a minta többször is jelen van. Ebben a kontextusban a Rabin-Karp algoritmus effektívebb keresést biztosít a szisztematikus szelekcióval. Ha egy keresett szövegrészletről tudjuk, hogy csak egyszer fordul elő a szövegben, akkor a „naiv" algoritmus hatékonyabb, mivel ebben az esetben a „naiv" algoritmus Big-O értéke O(M) és O(M*(N-M+1)) között mozog, míg a Rabin-Karp esetében a Big-O értéke O(1+M) és O(M*N) közötti lesz. A 'naive string-matching' algoritmus tehát effektívebb egyszeri egyezés kereséséhez, míg a Rabin-Karp a többször előforduló minták keresésében hasznosabb.

Zárógondolatok

Jelen tanulmányban a hashing elméleti alapjára és felhasználási területeire esett szó. Vázoltuk továbbá a 'naive string-match' és a Rabin karp" algoritmusokat is.

Köszönetnyilvánítás

A tanulmány a Kulturális és Innovációs Minisztérium és a Nemzeti Tehetség Program NTP-NFTÖ-22-A2-0249 pályázati azonosítószámú támogatásával valósult meg. A támogatást ezúton is tisztelettel köszönöm!

Irodalom

Zola, Andrew (2021): hashing. Megnyitás: 2023.06.18. Web: https://www.techtarget.com/searchdatamanagement/definition/hashing

Loo, Andrew (2023): Hash Function - An algorithm that converts a message into a hash value. Megnyitás: 2023.06.18. Web: https://corporatefinanceinstitute.com/resources/cryptocurrency/hash-function/

Net1. Megnyitás: 2023.06.18. URL: https://www.okta.com/au/identity-101/hashing-algorithm

Net2. Megnyitás: 2023.06.19. URL: https://stytch.com/blog/what-is-password-hashing/

Net3: Megnyitás: 2023.06.21. URL: https://www.geeksforgeeks.org/naive-algorithm-for-pattern-searching/

Net4: Megnyitás: 2023.06.22. URL: https://www.youtube.com/watch?v=nK7SLhXcqRo

Net5: Megnyitás: 2023.06.22. URL: https://www.programiz.com/dsa/rabin-karp-algorithm

Mark Burnett and James C. Foster (2004): Hacking the code. Oxford,United Kingdom: Syngress

Simplilearn (2023): Introduction to Big O Notation in Data Structure. Megnyitás: 2023.06.20. Web: https://www.simplilearn.com/big-o-notation-in-data-structure-article