

Review Article / Pregledni rad
Manuscript received: 2017-03-17
Revised: 2017-12-04
Accepted: 2017-12-11
Pages: 79 - 94

Zoran Hercigonja
Druga gimnazija Varaždin
Hrvatska
zoran.hercigonja@gmail.com

Sažetak: Internet nudi mogućnost pristupa mnogim sadržajima. Isto tako stvara mogućnost neovlaštenog preuzimanja sadržaja. Neovlašteno preuzeti sadržaj naziva se plagijatom. Za otkrivanje plagijata, poseže se vrlo često za računalnim softverima kao jednom obliku rješenja. Preduvjet kvalitetnog softvera je kvalitetan algoritam. U svijetu postoji mnogo algoritama za detekciju plagijata. U ovom radu su izdvojeni algoritmi s najboljim karakteristikama. U radu je ujedno dati pregled takvih algoritama s prikazom i usporedbom njihovih mogućnosti.

Keywords: Algoritam, Računalo, Plagijat, Softver, Detekcija

UVOD

Plagiranje kao hotimična ili nehotimična radnja, akt ili čin je vrlo aktualna tema. Pojedinac je neprestano izložen brojnim gotovim frazama, riječima, izrazima, rečenicama ili definicijama koje zaokupljaju njegovu pažnju i misao i „...htjeli, ne htjeli, imamo u ušima već gotove fraze svoje struke, pa se njima obilno služimo...“ [1]. Nove tehnologije kao što je na primjer Internet, osim što osiguravaju dostupnost sadržaja osiguravaju i prepisivanje u obliku direktnog kopiranja sadržaja. Učestalost plagiranja raste, jer većina poseže upravo za kopiranjem i kompilacijom tuđih radova. Sam razvoj i unapređenje informacijske tehnologije „...olakšava neovlašteno preuzimanje teksta...“ [2]. Razvoj tehnologije olakšava pristup sadržajima ali i prisvajanje frazeologije i odlomaka jednostavnom naredbom „copy/paste“ odnosno naredbom „kopiraj/zalijepi“. Iako su se razvojem moderne tehnologije riješili problemi vezani uz pisanje radova i dostupnost podataka, problem plagiranja neprestano je prisutan. Naravno paralelnim razvojem tehnološko-internetskih mogućnosti dostupnosti sadržaja i povećanja stope „literarne krađe“, razvili su se brojni softverski alati za otkrivanje i identificiranje postotaka udjela određenog sadržaja u nekom od radova kako bi se utvrdila stopa plagiranja. Sofisticirane metode plagiranja, stvorile su potrebu za stvaranjem sofisticiranih softvera za suzbijanje plagijata. Dakle „...Učestalost plagiranja je u porastu, razvoj informacijsko-komunikacijske tehnologije olakšava neovlašteno preuzimanje teksta, no istovremeno, zahvaljujući istoj tehnologiji, razvijaju se računalni programi i mrežne usluge za otkrivanje plagiranja...“ [2]. Razvoj brojnih tehnologija je omogućio stvaranje softvera za detekciju plagijata te na taj način ujedno pomogao razriješiti problem plagiranja. Stvaranje softverskog rješenja nije sasvim jednostavno. Potrebno je odrediti niz koraka temeljem kojih će softver pronaći plagijat odnosno krivotvorinu. Prije programiranja softvera, potrebno je osmisлити u pseudojeziku redosljed traženja plagiranih sadržaja. Takav niz koraka zove se algoritam. Tek izradom algoritma, može se planirati računalno odnosno softversko rješenje. No uz sve te ponuđene algoritme, potrebno je izdvojiti one koji su vrlo uspješni u utvrđivanju plagijata. Stoga se ovim radom daje pregled i usporedba te procjena kvalitete pojedinog algoritma u utvrđivanju plagijata na temelju njihovih osnovnih karakteristika.

ALGORITMI ANALIZE TEKSTOVA NA TEMELJU SLIČNOSTI ZNAKOVNIH NIZOVA

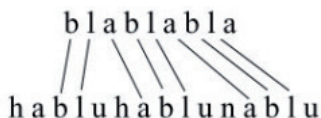
Gotovo svi softveri za realizaciju detekcije plagijata ostvareni su algoritmima analize tekstova na temelju sličnosti znakova. Većina softvera fizički uspoređuju riječ po riječ, rečenicu po rečenicu i odlomak po odlomak. Usporedba se ne provodi uvijek istim redosljedom to jest istim nizom koraka.

Analiza tekstova na temelju sličnosti nizova znakova podrazumijeva detaljan prolazak kroz svaku rečenicu nekog sadržaja te uspoređivanje uzorka sa zadanim sadržajem ko-

jem se utvrđuje razina plagiranja. Analiza teksta podrazumijeva korištenje dijelova ili cijelih uzoraka. Tako se istovremeno postiže brzina detektiranja, ali i dubina utvrđivanja sličnosti.

NAJDULJI ZAJEDNIČKI PODSLJED

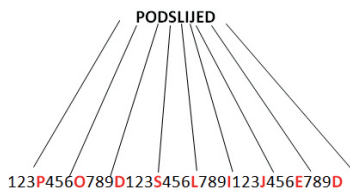
Najdulji zajednički podsljed podrazumijeva usporedbu znakova odnosno znakovnih nizova između dva paralelna teksta. Izrazito se naglašava da „...treba razlikovati ovakav algoritam od algoritma nalaženja najdužeg zajedničkog podniza...“[3]. Podsljed podrazumijeva prije svega praćenje duljine nekog slijeda da bi se na temelju toga mogla napraviti usporedba sa slijedom paralelnog teksta koji se uspoređuje s tekstem u obradi. Inicijalni smisao ovog algoritma sastoji se u „...traženju znakovnog niza unutar drugog i kao rezultat vraća duljinu podsljeda...“[3]. Ovim algoritmom traži se uzorak identificiran u tekstu. Tekst može biti jedna rečenica unutar koje se pokušava identificirati jedna riječ da bi temeljem toga mogla utvrditi sličnost s uspoređivanim tekstem. Ukoliko je odabrana jedna riječ kao uzorak: na temelju nje može se utvrditi cijela modificirana rečenica dodatnim (pridodanim) riječima. U modificiranoj rečenici s umetnutim riječima, odabirom jednog od uzoraka znakova (riječi) stvara se čvrsto pozicioniranje u toj rečenici. Dakle između odabranih uzoraka može biti „...proizvoljan broj umetnutih znakova...“[3]. Primjerice dva odabrana niza znakova: „blablably“ i „habluhablunablu“, potrebno je napisati jedan iznad drugoga kako bi se lakše utvrdila sličnost između znakova.



Slika 1: Uspoređivanje uzorka u podsljedu

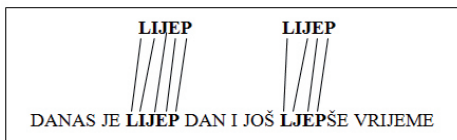
Prvi slijed znakova „blablably“ predstavlja uzorak na temelju kojega je provedena usporedba s drugim slijedom znakova „habluhablunablu“. Slijed znakova „blablably“ naziva se ujedno i podsljedom. Svaki uzorak je na neki način podsljed jer slova koja se pojavljuju u njemu, sadržana su u slijedu znakova kao u primjeru „habluhablunablu“. Povezivanje okomitim linijama povezuju se slova (znakovi) zajednička u oba slijeda znakova (u uzorku i originalnom slijedu znakova). Time se dolazi do ekvivalencije slova iz uzorka (prvog slijeda znakova) sa slovima drugog slijeda znakova. Takva uparena slova (znakovi) nazivaju se podsljedom. Radi očuvanja konzistentnosti prvotno izabranog i definiranog uzorka važno je da se znakovi pronalaze istim redoslijedom kako su bili definirani, a ne obrnutim redoslijedom.

Primjerice na slici 1, uzorak „blablably“ započinje tek s prvim slovom „b“ u originalnom slijedu, iako je moguće prije slova „b“ prepoznati i znak „a“ koji se također nalaži definiran u uzorku. Prateći uvijek isti redoslijed definiranog uzorka, konzistentno se identificira uzorak u originalnom slijedu znakova. Primjerice kao uzorak uzima se riječ „podsljed“, a kao originalni slijed znakova dodaje se proizvoljan broj znakova je u ovom primjeru nadopunjen brojevima.



Slika 2: Traženje podsljeda

Pronađen je podsljed neovisno o tome što je originalni slijed znakova nadopunjen brojkama. Kada u dva slijeda znakova oba započinju istim slovom, najsigurnije je spojiti ta dva slova kao dio podsljeda. Ukoliko se prvo slovo kao na slici 1 nalazi na nekom desnijem mjestu (primjerice na desnijem mjestu je bilo slovo „b“ kao prvo slovo uzorka) linija kojom se spajaju ta dva slova u oba niza se može preusmjeriti u lijevo bez da se uzrokuje presijecanje linija. To je najsigurniji način utvrđivanja da se radi o prvom slovu. U nekim se situacijama mogu prva slova razlikovati. Nemoguće je da će oba slova biti dio podsljeda, nego će biti potrebno bar jedno od njih (ili oba) ukloniti. Konačan slikoviti primjer upotrebe ovog algoritma može se provjeriti primjerom traženja i usporedbe uzorka: „lijep“ u rečenici „Danas je lijep dan i još ljepše vrijeme.“ Ovaj primjer je istovjetan primjeru na slici 2., jer se uzorak „lijep“ nadopunjuje dodatnim znakovima „Danas je...dan i još ljepše vrijeme.“



Slika 3: Traženje podsljeda u konkretnoj rečenici

Uzorak je identificiran na dva mjesta iako u riječi „ljepše“ nije moguće utvrditi cjeloviti uzorak, ali su identificirana prva slova uzorka „lijep“. Podsljed i podniz su međusobno povezani iako su u određenim aspektima različiti, ali jedno bez drugoga bi u softverima za detekciju plagijata izazvalo brojne propuste, što opet rezultira nedostatnom identifikacijom plagijata.

NAJDULJI ZAJEDNIČKI PODNIZ

Podniz i podsljed su intuitivno vrlo srodni pojmovi ali kako je već naglašeno „...treba razlikovati ovaj algoritam od algoritma nalaženja najdužeg zajedničkog podniza...“ [3]. Prema tome ovaj algoritam razlikuje se od algoritma najdulji zajednički podsljed po tome što se pod pojmom podniz smatra „...skup znakova iz originalnog znakovnog niza koji je povezan, dok se podsljedom smatra bilo koji podskup znakova iz originalnog niza koji se može dobiti brisanjem nula ili više znakova originalnog niza...“ [3]. S obzirom na navedeno, podniz je definiran kao neprekinuti niz originalnih znakova koji se u istom redoslijedu mora pronaći u drugom tekstu ili nizu. Kod podsljeda u originalni niz znakova može se dodati ili izbrisati proizvoljan broj znakova. Na primjer imamo originalni

niz „banana“. Odbacivši sufikse i prefikse niza „banana“, podniz bi izgledao ovako „ana“. Za isti niz „banana“ odbacivanjem jednog, nijednog ili više ne nužno uzastopnih znakova, podsljed bi glasio „baaa“. Važno je za naglasiti da se u svrhu ovog algoritma koristi takozvano sufiksno stablo. Prema tome stablo sufiksa je struktura podataka koja omogućava rješavanje raznih problema vezanih uz znakovne nizove u linearnom vremenu. Ako je znakovni niz označen sa $str = t_1 t_2 t_3 \dots t_n$ onda je $T_i = t_i \dots t_n$ sufiks od str koji počinje na poziciji $i \dots$

Kao primjer može poslužiti riječ mississippi. Algoritam za zadanu riječ izdvaja sve moguće podnizove te iste riječi koje se kasnije sortiraju prema početnom slovu. Na slici 4 je prikazano početno raščlanjivanje riječi mississippi po parametrima T_1 - T_{12} . Nakon raščlanjivanja, provodi se sortiranje po početnom slovu nazvanom sufiks. Za primijetiti je da sortiranjem prefikse po slovima, neki od njih poprimaju zajedničke prefikse (npr. i, p, s).

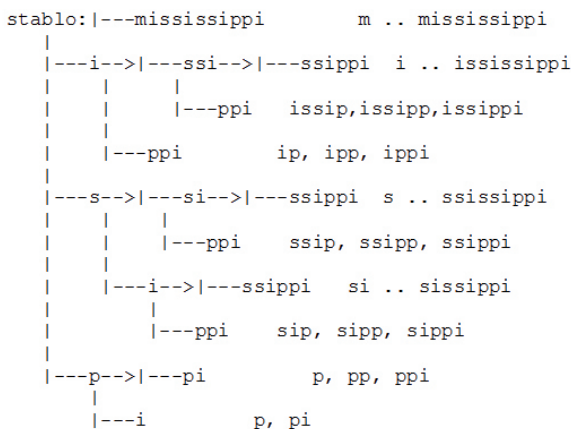
Sljedeći korak je izrada stabla sufiksa tako da se sufiksi sa zajedničkim prefiksom interpretiraju kao korijen u stablu. Točnije sufiksi sa zajedničkim prefiksom imaju zajednički korijen u stablu.

- $T_1 = mississippi = str$
- $T_2 = ississippi$
- $T_3 = ssissippi$
- $T_4 = sissippi$
- $T_5 = issippi$
- $T_6 = ssippi$
- $T_7 = sippi$
- $T_8 = ippi$
- $T_9 = ppi$
- $T_{10} = pi$
- $T_{11} = i$
- $T_{12} = (\text{prazno})$

Slika 4: Početno raščlanjivanje riječi mississippi na podnizove (preuzeto iz [4])

- $T_{11} = i$
- $T_8 = ippi$
- $T_5 = issippi$
- $T_2 = ississippi$
- $T_1 = mississippi$
- $T_{10} = pi$
- $T_9 = ppi$
- $T_7 = sippi$
- $T_4 = sissippi$
- $T_6 = ssippi$
- $T_3 = ssissippi$

Slika 5: Sortiranje prefiksa po slovima (preuzeto iz [4])

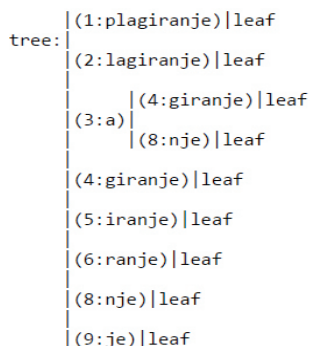


Slika 6: Sufiksno stablo riječi mississippi (preuzeto iz [4])

Najduži zajednički podniz traži se pomoću generaliziranog stabla sufiksa. Generalizirano stablo sufiksa sadrži sve sufikse više znakovnih nizova (izvornih tekstova programa)

koji se uspoređuju. „...Čvorovi takvog stabla moraju biti označeni ovisno da li pripadaju prvom, drugom ili oba osnovna niza...“[3]. Dakle „...najveći zajednički podniz će biti najdublji čvor koji je označen kao dio oba osnovna niza[3]. Prema tome u slučaju riječi „mississippi“ zajednički korijeni su nad prefiksima **i,p,s**. Primjena ovog algoritma je moguća i u određivanju sličnosti izvornih tekstova ako je izgrađeno općenito sufiksno stablo koje će sadržavati sve sufikse oba teksta. Kako bi usporedba bila moguća potrebno je voditi računa da se za svaki čvor stabla zna kojem tekstu pripada: prvom, drugom ili objema tekstovima.

Raščlanjivanjem niza znakova „plagiranje“, nastat će situacija kao na slici 7. sufiksno stablo riječi „plagiranje“.



Slika 7: Sufiksno stablo riječi „plagiranje“

Za primijetiti je da riječ plagiranje ima samo dvije grane od kojih je najdublji čvor (9:je). Može se zaključiti da je najdulji zajednički podniz „je“ prema kojem će se vršiti daljnje pretraživanje u uspoređivanju nizova znakova. Identifikacija plagijata pomoću softvera ne bi bila potpuna bez postojanja algoritama koji „mjere“ udaljenost u podnizovima i podsljedovima.

HAMMINGOVA UDALJENOST

Hammingova udaljenost kao algoritam za utvrđivanje plagijata upotrebljava se u teorijama informacija gdje je udaljenost dvaju nizova znakova iste duljine definirane kao broj bitova ili lokacija na kojima ta dva postojeća niza ne sadrže identične znakove. Detekcija plagijata temelji se na prepoznavanju modifikacije originalnih znakova. To dokazuje definicija Hammingove udaljenosti: „...Hammingova udaljenost (HD) između dvije riječi je broj bitova u kojima se one razlikuju...“[5]. U nastavku plagijat se utvrđuje na temelju pogrešaka u bitovima odnosno lokacijama. Dakle „...ako dvije riječi imaju Hammingovu udaljenost x , tada je potrebno pogriješiti x bitova da bi se jedna kodna riječ pretvorila u drugu...“ [5].

Za primjer je uzeta riječ pretvorena u bitove. Ovo su samo simbolični nazivi riječi koje su pretvorene u binarni kod.

Primjer 1(Riječi u binarnom kodu):**riječ 1: 01100110****riječ 2: 00101001**

Plavom bojom su prikazani originalni bitovi riječi 1, a crvenom su označene „pogreške“ odnosno promijenjene lokacije bitova. Dakle Hammingova udaljenost između riječi 1 i riječi 2 je pet bitova ili pet pogrešaka lokacije. Algoritam Hammingove udaljenosti doslovno, mjeri minimalan broj supstitucija ili zamjena koje su potrebne za promjenu jedne riječi u drugu riječ ili broj grešaka prilikom transformacije jedne riječi u drugu. Osnovni uvjet Hammingove udaljenosti je da nizovi znakova moraju imati nužno isti broj elemenata što naposljetku implicira da se dogodila samo zamjena, a ne i brisanje ili do-davanje kao kod drugih algoritama.

Primjer 2 (Hammingova udaljenost riječi „Bijeg“ i „Tijek“):**Bijeg****Tijek**

Hammingova udaljenost između ove dvije riječi jednaka je ukupno dvije lokacije ili dvije pogreške odnosno dva bita. Promijenjene su pozicije riječi **B** u **T** i **g** u **k**. Ovo je prikaz jedne mogućnosti primjene mjerenja udaljenosti, no postoji i još nekoliko oblika bržeg i efikasnijeg načina mjerenja udaljenosti kao što je Levensteinova udaljenost.

LEVENSTEINOVA UDALJENOST

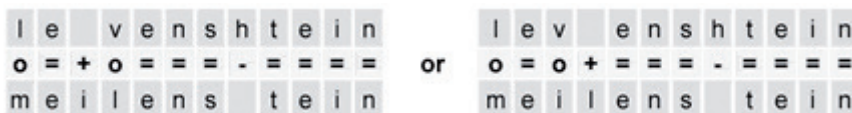
Levensteinova udaljenost „...izračunava najmanji broj operacija koje su potrebne za transformaciju jedne riječi u drugu odnosno transformaciju jednog znakovnog niza u drugi...“[6]. Levensteinova udaljenost definira se kao broj potrebnih akcija s kojima se jedan znakovni niz pretvara u drugi pri čemu se zadovoljava mogućnost plagijatora da mijenja, dodaje ili briše znakove. Rabi se matrica izračunavanja akcija tipa (m,n) gdje se zamjenjuje m-prefiks s n-prefiksom.

		m	e	i	l	e	n	s	t	e	i	n
	0	1	2	3	4	5	6	7	8	9	10	11
l	1	1	2	3	3	4	5	6	7	8	9	10
e	2	2	1	2	3	3	4	5	6	7	8	9
v	3	3	2	2	3	4	4	5	6	7	8	9
4	4	4	3	3	3	4	4	5	6	6	7	8
n	5	5	4	4	4	4	3	4	5	6	7	7
s	6	6	5	5	5	5	4	3	4	5	6	7
h	7	7	6	6	6	6	5	4	4	5	6	7
t	8	8	7	7	7	7	6	5	4	5	6	7
e	9	9	8	8	8	7	7	6	5	4	5	6
i	10	10	9	8	9	8	8	7	6	5	4	5
n	11	11	10	9	9	9	8	8	7	6	5	4

Slika 8: Levensteinova matrica izračunavanja udaljenosti (preuzeto iz [6])

Za primijetiti je da se s lijeve strane riječ „Levenshtein“ nalazi zapisana u stupcu s n-prefiksom, a riječ „meilenstein“ s m-prefiksom u retku matrice. Matrica može biti popun-

jena s gornje lijeve strane prema donjem desnom kutu. Svaka vodoravna ili okomita aktivnost transformacije odgovara akciji brisanja ili dodavanja. Svaka promjena znaka označava se brojem transformacija u obliku brisanja ili dodavanja. U prikazanoj matrici na slici 9 nazire se dijagonalna promjena što znači da je baš svaki znak početne riječi „Levenshtein“ podvrgnut transformacijama.



Slika 9: Mogući putevi kroz matricu (preuzeto iz [6])

Putovi zamjene riječi „Levenshtein“ mogu se svesti na riječ „meilenshtein“ ili riječ „meilenshtein“. Sve ovisi o potrebi i interesu plagijatora.

Primjer 3 (Levensteinova udaljenost znakova „lijek“ i „riječi“):

lijek -> rijek
rijek->riječ
riječ->riječi

Levensteinova udaljenost znakova „lijek“ i „riječi“ iznosi tri. Dakle napravljene su točno tri promjene da bi se znak lijek pretvorio u riječi. Prva promjena je podrazumijevala zamjenu početnih slova „l“ slovom „r“. Drugi korak zamjene je podrazumijevao zamjenu posljednjeg znaka „k“ u znak „č“. I u trećem koraku je nad znakom „riječ“ dodan znak „i“, čime je znak „lijek“ potpuno transformiran u znak „riječi“. Zbog utvrđenih propusta u radu, ovaj je algoritam modificiran do razine Damerau-Levensteinove udaljenosti.

DAMERAU-LEVENSTEINOVA UDALJENOST

Damerau-Levensteinova udaljenost predstavlja proširenje postojeće Levensteinove udaljenosti. Definicija Levensteinove udaljenosti podrazumijeva: Levensteinova udaljenost „...izračunava najmanji broj operacija koje su potrebne za transformaciju jedne riječi u drugu odnosno transformaciju jednog znakovnog niza u drugi...“ [6]. Damerau-Levensteinova udaljenost potom podrazumijeva „...dodavanje akcije kojom se lokacije dva susjedna znaka jednog niza mogu zamijeniti...“ [7]. S time se može iznijeti pretpostavka da su riječi nekog teksta kratke te da je broj pogrešaka prilikom pisanja rijetko veći od dvije lokacije ili pogreške.

Primjer 4 (Damerau-Levensteinova udaljenost):

tuorka->utorka
utorka->utorak

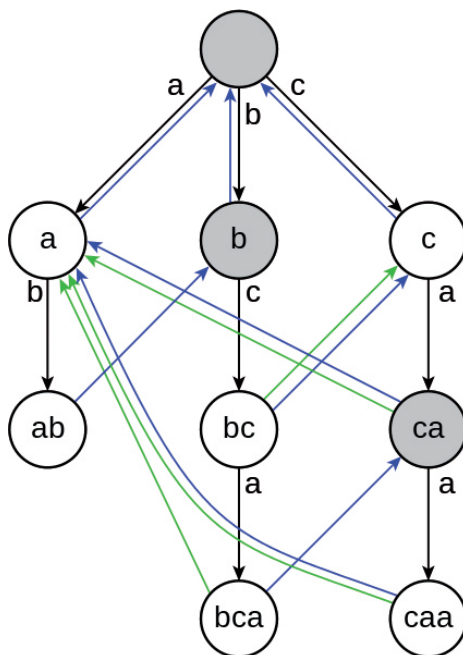
Zamjena susjednih znakova dogodila se na prefiksima riječi „tuorka“ i „utorka“ u jednom slučaju. Navedena promjena, broji se kao zamjena jedne lokacije ili identifikacija jedne pogreške. Druga zamjena se odnosi na riječi „utorka“ i „utorak“ gdje se zamjena susjednih znakova dogodila na kraju riječi to jest na sufiksima. Pritom je identificirana

jedna lokacija ili pogreška. Stoga valja zaključiti da se rade maksimalno dvije pogreške prilikom pisanja. Naravno ne bi trebalo zanemariti mogućnost da se može pojaviti i neka druga varijanta pogreške ili zamjene što bi značilo da je početna pretpostavka Damerau-Levensteinove udaljenosti zamjene lokacija dva susjedna znaka relativna. Algoritmi su se pokazali vrlo uspješnima u radu.

AHO-CORASICK ALGORITAM

Aho-Corasick algoritam je „...klasično i skalabilno rješenje za određivanje točnog podudaranja znakova te naširoko poznat algoritam za pronalaženje više uzoraka...” (Vidanagamachchi i sur, 2012). Implementirali su ga Alfred V. Ahoa i Margaret J. Corasick. Pretpostavka za rad ovog algoritma je poznato stablo sufiksa korišteno u algoritmima analize tekstova na temelju sličnosti znakovnih nizova. Primjer ovakvog algoritma može se prezentirati pomoću jednostavnog uzorka (bca).

Primjer 5 (Primjena Aho-Corasick algoritma s [bca] uzorkom)



Slika 10: Aho-Corasick algoritam traženja otisaka izvornih tekstova (preuzeto iz [8])

Dakle struktura podataka ima po jedan čvor za svaki prefiks svakog znaka. Primjenom uzorka (bca) kreirati će se čvorovi (bca), (bc), (b) i (). Kreiranjem tih čvorova, vidljivo je da se sufiksno stablo sastoji od crne usmjerene grane i plave usmjerene grane.

Samim time postoji crna grana od (bc) do (bca) uzorka (bca). Zatim postoji plava usmjerena sufiks grana od svakog čvora do čvora koji je najdulji mogući striktni sufiks tog čvora u grafu. U navedenom primjeru za čvor (caa) striktni sufiksi su (aa), (a) i (). Najdužim

striktnim sufiksom u grafu od (aa), (a) i () je (a), tako da postoji plava grana od (ca) do (a). Isto tako postoji i zelena grana sufiksa od svakog čvora do sljedećeg čvora do kojeg se može doći prateći plave grane. Primjerice, postoji zelena grana od (bca) do (a) zbog toga što je (a) prvi čvor do kojeg se dolazi putem plave grane do (ca), a onda do (a). Na svakom koraku, trenutni čvor se produljuje tražeći vlastito dijete, a ako ono ne postoji, onda algoritam nalazi sufiks. Kada algoritam dođe do čvora izbacuje sve moguće ulaze koji završavaju na trenutnoj poziciji karaktera u unesenom tekstu. Ovaj algoritam je takoreći dobar. No kvalitetu algoritma moguće je utvrditi tek nakon usporedbe njegove funkcionalnosti s drugim algoritmima.

BOYER-MOORE ALGORITAM

Boyer-Moore algoritam „...uspoređuje simbol uzorka i simboli teksta s desna na lijevo, počevši od zadnjeg simbola uzorak...“[9]. Razvili su ga Robert S. Boyer i J. Stroter Moore. Princip rada ovog algoritma temelji se na poravnavanju uzoraka sa zadanim tekstem. To znači da se uzorak doslovno pomiče unutar teksta s ciljem ostvarivanja podudaranja. Boyer-Moore algoritam koristi informacije dobivene iz pretprocesiranja kako bi preskočio što više poravnanja te kako bi se postigla odgovarajuća brzina prolaženja kroz tekst kroz utvrđena podudaranja.

Primjer 6 (Primjena Boyer-Moore algoritma, preuzeto iz [10])

U prvom koraku zadan je niz znakova nekog teksta

G C A T C G C A G A G A G T A T A C A G T A C G

i uzorak

G C A G A G A G

Prvi pokušaj

G C A T C G C A G A G A G T A T A C A G T A C G

1

G C A G A G A G

Budući da funkcioniranje algoritma zahtjeva kretanje od lijevog prema desnom kraju, prvotni položaj uzorka mora biti smješten skroz lijevo. Za početak nema podudaranja.

Drugi pokušaj

G C A T C G C A G A G A G T A T A C A G T A C G

3 2 1

G C A G A G A G

Pomicanjem za jedan korak dalje, dogodilo se prvo podudaranje slova A. Algoritam je nastavio dalje pomicati uzorak prema desnom kraju i dolazi do podudarnosti slova A i G. Sveukupno su napravljena tri koraka pomicanja u desnu stranu.

Treći pokušaj

G C A T C G C A G A G A G T A T A C A G T A C G
 8 7 6 5 4 3 2 1
 G C A G A G A G

U trećem pokušaju, uzorak je pomaknut sveukupno osam koraka u desnu stranu i dobiveno je prvo cjelovito podudaranje. Naravno postupak još nije gotov jer algoritam identificira još jednu podudarnost. Budući da je ova podudarnost ostala zapamćena, algoritam se više neće vraćati na nju, nego će je preskočiti.

Četvrti pokušaj

G C A T C G C A G A G A G T A T A C A G T A C G
 3 2 1
 G C A G A G A G

U četvrtom pokušaju, pronađena su dva nova podudaranja slova A i G. Algoritam se pomiče za tri mjesta u desnu stranu. No podudarnost cjelovitog uzorka nije postignuta. Algoritam bilježi djelomičnu podudarnost uzorka s nizom znakova teksta kako se više ne bi vraćao na njega.

Peti pokušaj

G C A T C G C A G A G A G T A T A C A G T A C G
 2 1
 G C A G A G A G

Peti i ujedno posljednji pokušaj podrazumijeva podudaranje slova G u uzorku što znači da je uzorak pomaknut za dodatna dva mjesta u desno. Cjelovito podudaranje nije postignuto, ali su identificirana mjesta u tekstu gdje je prepoznati dio zadanog uzorka. U odnosu na prethodni algoritam, Boyer-Moore algoritam je prosječno brz u radu, ali i praktično koristan pogotovo ako se upotrebljava u kombinaciji s algoritmima s podslijedom ili podnizom. Naravno praktičnost nije upitna, već je upitna brzina rada u usporedbi sa sljedećim Rabin-Karp algoritmom.

RABIN-KARP ALGORITAM

Algoritam Rabin-Karp razvili su Michael O. Rabin i Richard M. Karp 1987. godine. Algoritam omogućava pretraživanje znakovnih nizova „...koristeći funkciju raspršenja...“ [10]. Algoritam je poznat po svojoj iznimnoj brzini rada. Brzina Rabin-Karp algoritma

temelji se na brzom uspoređivanju znakovnih nizova. Osnovna ideja je korištenje sažetaka k-grama. Zbog toga je glavna ideja usporediti sve sažetke k-grama. Pod k-gramima se podrazumijevaju parovi znakova koji se međusobno uspoređuju. To se radi na način da se izračunaju sve te vrijednosti za duge znakovne nizove unutar kojih se traži neki podniz. Deklaracija algoritma je sljedeća [10]:

Neka je k-gram k-znamenasti broj $c_1 \dots c_k$ u nekoj bazi b . Kao funkcija računanja sažetka se uzima:

$$H(C_1 \dots C_k) = C_1 b^{(k-1)} + C_2 b^{(k-2)} + C_3 b^{(k-3)} + \dots + C_k$$

Za izračunavanje vrijednosti sažetka novog k-grama potrebno je postaviti sljedeći račun:

$$H(C_1 \dots C_{(k+1)}) = (H(C_1 \dots C_k) - C_1 b^{(k-1)})b + C_{(k+1)}$$

S obzirom na to da je b^{k-1} konstanta, svaka iduća vrijednost se računa pomoću dvije operacije zbrajanja, i dvije operacije množenja. Te operacije se uzimaju kao „modulo“ neke vrijednosti, najčešće najveća vrijednost za cijeli broj. Da bi se pronašao željeni uzorak unutar nekog znakovnog niza, Rabin-Karp algoritam putem deklarirane relacije izračunava vrijednost trenutnog detektiranog uzorka koja se uspoređuje s vrijednosti početnog zadanog uzorka. Ukoliko se vrijednosti poklapaju, radi se dodatna provjera ASCII koda oba uzorka i u slučaju poklapanja, pronađen je željeni uzorak. U suprotnom algoritam prolazi dalje znakovnim nizom i uspoređuje vrijednosti relacija. Algoritam završava s radom tek kada dođe do kraja znakovnog niza neovisno o tome da li je već pronašao i detektirao željeni uzorak.

Funkcionalnost Rabin-Karp algoritma može se sagledati kroz konkretan primjer.

Primjer 7 (Primjena Rabin-Karp algoritma)

Za znakovni niz uzeto je jedanaest ne uzastopno slijednih brojeva. To su brojevi: **31415926535**. Kao uzorak pretraživanja uzet je **broj 26**. Duljina znakovnog niza je **11** brojeva. Postupak je vođen izvornom notacijom formule $P \times \text{mod } q$ pri čemu P predstavlja par brojeva koji ulaze u uzorak usporedbe, zatim q koji se interpretira kao duljina znakovnog niza. Konačna deklaracija početnih veličina izgleda ovako:

Niz = 31415926535

P = 26

q = 11

$P \times \text{mod } q \rightarrow 26 \times \text{mod } 11 = 4$

Svaki par brojeva u zadanom nizu će sustavno prolaziti kroz formulu $P \times \text{mod } q$ i rezultat će se uspoređivati s dobivenom vrijednosti 4, dobivenom kao rezultat relacije $26 \times \text{mod } 11 = 4$. Kao konačno rješenje će se uzeti broj koji se poklapa s rezultatom relacije $26 \times \text{mod } 11 = 4$.

Princip rada algoritma kreće s lijeve strane znakovnog niza. U relaciju ulaze prva dva broja 3 i 1. Njihov rezultat će se usporediti s rezultatom početne relacije.

3	1	4	1	5	9	2	6	5	3	5
---	---	---	---	---	---	---	---	---	---	---

$$31 \times \text{mod } 11 = 9 \neq 4$$

Rezultat provedene relacije je 9 što se ne podudara s vrijednosti 4. To znači da je potrebno prjeći na idući par brojeva. Sljedeći par brojeva je 1 i 4.

3	1	4	1	5	9	2	6	5	3	5
---	---	---	---	---	---	---	---	---	---	---

$$14 \times \text{mod } 11 = 3 \neq 4$$

Ni ovaj rezultat nije zadovoljavajuć. Vrijednost relacije je 3. Provodi se daljnja usporedba idućih parova brojeva.

3	1	4	1	5	9	2	6	5	3	5
---	---	---	---	---	---	---	---	---	---	---

$$41 \times \text{mod } 11 = 8 \neq 4$$

3	1	4	1	5	9	2	6	5	3	5
---	---	---	---	---	---	---	---	---	---	---

$$15 \times \text{mod } 11 = 4 = 4$$

U ovom slučaju dobilo se poklapanje vrijednosti dobivene kao rezultat relacije 4. No uzorak 15 se ne poklapa s traženim uzorkom 26. Algoritam Rabin-Karp na temelju izračunate vrijednosti relacije pronalazi potencijalna mjesta u nizu znakova koja bi mogla odgovarati prvotnom zadanom uzorku. No to ne znači da će vrijednost kao rezultat relacije garantirati identičnost uzorka. Stoga Rabin-Karp algoritam mora napraviti dodatnu provjeru znakova. Točnije provjerava ASCII kod znakova niza. Primjerice ASCII kodovi za brojeve 1 i 5 su različiti od brojeva 2 i 6. Postupak rada algoritma se nastavlja na sljedećem paru brojeva.

3	1	4	1	5	9	2	6	5	3	5
---	---	---	---	---	---	---	---	---	---	---

$$59 \times \text{mod } 11 = 4 = 4$$

Dogodila se ista situacija kao u prethodnom koraku. Budući da nema podudaranja uzorka, rad algoritma se provodi dalje.

3	1	4	1	5	9	2	6	5	3	5
---	---	---	---	---	---	---	---	---	---	---

$$92 \times \text{mod } 11 = 4 = 4$$

Ponovno je došlo do podudaranja po rezultatu vrijednosti relacije, ali ne i prema usporedbi znakova putem ASCII koda.

3	1	4	1	5	9	2	6	5	3	5
---	---	---	---	---	---	---	---	---	---	---

$$26 \times \text{mod } 11 = 4 = 4$$

Dobivene vrijednosti relacija trenutnog uzorka i početnog zadanog uzorka se podudaraju. Isto tako ASCII kod znakova 2 i 6 su identični i došlo se do detekcije početnog uzorka. Iako je uzorak pronađen i identificiran, algoritma nastavlja dalje s radom dok ne dođe do kraja znakovnog niza.

3	1	4	1	5	9	2	6	5	3	5
---	---	---	---	---	---	---	---	---	---	---

$$65 \times \text{mod } 11 = 10 \neq 4$$

Vrijednosti izračunatih relacije nisu podudarne. Nema potrebe za daljnjom usporedbom ASCII koda znakova.

3	1	4	1	5	9	2	6	5	3	5
---	---	---	---	---	---	---	---	---	---	---

$$53 \times \text{mod } 11 = 9 \neq 4$$

3	1	4	1	5	9	2	6	5	3	5
---	---	---	---	---	---	---	---	---	---	---

$$35 \times \text{mod } 11 = 2 \neq 4$$

U posljednja tri koraka nije bilo nikakvog podudaranja. Algoritam je završio s radom kada je došao do kraja znakovnog niza. Početni uzorak zadan na početku je pronađen i identificiran prema broju izračunate vrijednosti relacije te dodatno provjeren putem usporedbe ASCII koda početnog uzorka i trenutnog uzorka detektiranog u znakovnom nizu. Algoritam je vrlo brz s obzirom na dvostruku provjeru koju radi prilikom detekcije uzorka.

ZAKLJUČAK

U ovom radu razjašnjeni su svi načini funkcioniranja sustava za detekciju plagijata preko algoritama i njihovog funkcioniranja. Napravljen je pregled i zaokružena cjelina funkcioniranja jednog složenog mehanizma kao što je softver za detekciju plagijata. Kvalitetan softver za računalnu detekciju plagijata kvalitetan je u tolikoj mjeri u kojoj je kvalitetno osmišljen algoritam za detekciju plagijata. Naravno dosadašnji algoritmi rješavali su dorasle probleme detekcije plagijata. No novi izazovi i vrlo kreativni načini plagiranja, zahtijevaju poboljšanje postojećih algoritama ili izradu novih na temelju njihovih kombinacija. Pritom je potrebno kombinirati najbolje mogućnosti koje mogu algoritmi ponuditi. Primjerice navedeno je da je Boyer-Moore algoritam prosječno brz u radu, ali i praktično koristan pogotovo ako se upotrebljava u kombinaciji s algoritmima s podsljedom ili podnizom. Dakle budućnost računalne detekcije plagijata to jest de-

tekncije pomoću računalnog softvera zavisi o kvalitetnim i poboljšanim algoritmima. Za poboljšanje algoritama, potrebno je praćenje evolucije trendova plagiranja i na taj način stvarati kreativna algoritamska rješenja.

LITERATURA

- [1] Raos N (2014) Što je to plagijat?, ViewPoint Dostupno na [http://hrcak.srce.hr/index.php?show=clanak&id_clanak_jezik=173199] (preuzeto 10.01.2017.)
- [2] Baždarić K, Pupovac V, Zulle L, Petrovečki M (2009) Plagiranje kao povreda znanstvene i akademske čestitosti, Dostupno na <http://hrcak.srce.hr/38691> (preuzeto 10.01.2017.)
- [3] Design and Analysis of Algorithms, opis algoritama za pronalaženje najduljeg zajedničkog podslijeda Dostupno na <http://www.ics.uci.edu/~eppstein/161/960229.html> (preuzeto 10.01.2017.)
- [4] L. Allison, članak Suffix trees, Dostupno na <http://www.allisons.org/ll/AlgDS/Tree/Suffix/> (preuzeto 10.01.2017.)
- [5] Nikolić B (1987) Jednostavna metoda za analizu promjena na jednom entitetu opisanom nad skupom kvalitativnih varijabli [<http://hrcak.srce.hr/108291>] (preuzeto 13.01.2017.)
- [6] The Levenshtein Algorithm (2012) Dostupno na <http://www.levenshtein.net/> (preuzeto 13.01.2017.)
- [7] Damerau F (1964) A technique for computer detection and correction of spelling errors, Communications of the ACM
- [8] Algorithm of the Week: Aho-Corasick String Matching Algorithm (2013) Dostupno na <http://architects.dzone.com/articles/algorithm-week-aho-corasick> (preuzeto 13.01.2017.)
- [9] Melichar B (2006) Text searching algorithms Dostupno na <http://www.stringology.org/athens/TextSearchingAlgorithms/tsa-lectures-2.pdf> (preuzeto 13.01.2017.)
- [10] Cho C, Lee S, Tan C, Tan Y (2004) Network Forensics on Packet Fingerprints, dostupno na http://link.springer.com/chapter/10.1007%2F0-387-33406-8_34#page-1 (preuzeto 13.01.2017.)

Algorithms for computer plagiarism detection

Abstract: Internet offers the possibility of access to many amenities. It also creates the possibility of unauthorized downloading content. Unauthorized downloadable content is called plagiarism. For plagiarism detection, we use the computer software. Precondition for quality software is a quality algorithm. In the world there are many algorithms to plagiarism detection. This study focuses on the algorithms with the best characteristics and functionality. The study gives an overview of algorithms and their capabilities.

Keywords: Algorithm, Computer, Plagiarism, Software, Detection

List of figures:

Figure 1: Comparison of the sample in subsection

Figure 2: Searching for subsection

Figure 3: Finding a subsection in the concrete sentence

Figure 4: Initial breakdown of mississippi words into subheadings

Figure 5: Sort the prefix by letter

Figure 6: Suicidal tree word mississippi

Figure 7: Suffix tree word "plagiarism"

Figure 8: Levenstein's distance calculation matrix

Figure 9: Possible paths through the matrix

Figure 10: Aho-Corasick algorithm for searching for prints of original texts

List of examples:

Example 1 (Binary Code Words)

Example 2 (Hamming's distance to the word "escape" and "flow")

Example 3 (Levenstein's Distance of the Characters "Cure" and "Words")

Example 4 (Damerau-Levenstein's Distance)

Example 5 (Application of Aho-Corasick algorithm with [bca] sample)

Example 6 (Application of Boyer-Moore algorithm,)

Example 7 (Application of Rabin-Karp algorithm)