Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 007.51
DOI: https://doi.org/10.24867/07BE48Trkulja

IMPLEMENTACIJA I TESTIRANJE ALGORITAMA ZA TAČNO PODUDARANJE STRINGOVA

IMPLEMENTATION AND TESTING OF STRING MATCHING ALGORITHMS

Branislav Trkulja, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu su opisani algoritmi koji su najčešće u upotrebi za tačno podudaranje stringova. Razmatrani su "Brute force" metoda, algoritam Rabin-Karp, Knuth-Morris-Pratt i Boyer Moore algoritam. Nabrojani algoritmi su implementirani u C# programskom jeziku i izvršena su testiranja njihovih performasi. Rad sadrži rezultate dobijene testiranjem nad dve vrste testnih podataka. Jedna vrsta podataka su veštački generisani primeri koji treba da istaknu dobre i loše strane algoritama, dok drugu vrstu podataka čine realni primeri: tekst knjige „Rat i mir" i CIM/XML fajl sa 2GB podataka.*

Ključne reči: *Pretraga teksta, Knuth-Morris-Pratt algoritam, Boyer Moore algoritam, Rabin Karp algoritam*

Abstract – *This paper describes the most commonly used algorithms for exact string matching. The Brute force method, the Rabin-Karp algorithm, the Knuth-Morris-Pratt algorithm and the Boyer Moore algorithm are discussed. The above algorithms were implemented in C # programming language and their performance was tested. The paper contains the results obtained by testing over two types of test data. One type of data is artificially generated examples that should highlight the good and bad sides of algorithms, while the other type of data is real examples: a 3226571 character text book and CIM / XML file with 2GB data.*

Keywords: *Text searching algorithms, Knuth-Morris-Pratt algorithm, Boyer Moore algorithm, Rabin Karp algorithm*

1. UVOD

U računarstvu algoritmi podudaranja stringova su važna klasa string algoritama koji pokušavaju da nađu mesto (ili više mesta) na kome se unutar teksta nalazi traženi obrazac.

Tipično, obrazac i tekst se sastoje od niza karaktera, gde tekst može biti ceo dokument, a obrazac može biti jedna reč, njen deo ili tekstualni fragment. Obično je tekst nastao iz nekog prirodnog jezika, ali to može biti bilo koja vrsta binarnih podataka u računaru.

Kada se takav binaran podatak pretražuje mogu se primeniti algoritmi podudaranja stringova, čime oni dobijaju više na značaju.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Aleksandar Erdeljan, red. prof.

Problem sa pretragom teksta jeste pronaći svako podudaranje obrasca u velikom tekstu. Podudaranje predstavlja da su svi karakteri obrasca sadržani u tekstu u istom redosledu. Ovaj problem se jasno javio u sedamdesetim godinama prošlog veka, kada su kompanije koje programiraju počele proizvoditi softversku obradu teksta i povećale potrebu za pronalaženjem brzih i efikasnih rešenja za ovaj problem. Takođe je jasno da je potraga za tekstom osnova postupka pronalaženja informacija u mnogim poljima, kao što su baze podataka, pretrage sadržaja knjiga, kao i pretraga web sadržaja.

Postojeći algoritmi pretrage su razvrstani u dve kategorije: algoritmi tačnog podudaranja i algoritmi približnog podudaranja. Neki od algoritama tačnog podudaranja su *Brute force*, Knuth-Morris-Pratt, Boyer-Moore i Rabin Karp algoritam. Algoritami približnog podudaranja omogućavaju da odredimo ograničen broj različitih znakova u podudaranjima.

Ovaj rad se bavi gore nabrojanim algoritama tačnog podudaranja i ima za cilj da testira i pokaže primenu tih algoritama za pretragu velikih tekstualnih fajlova na osnovu unetog obrasca. Takođe, analizirana je složenost algoritama i rad pokušava da pomogne pri izboru algoritma za odredjenu pretragu.

U poglavlju 2. će biti opisani poznati algoritmi za pretragu. U poglavlju 3. će biti prikazani rezultati testiranja algoritama za poduaranje obrazaca.

2. POZNATI ALGORITMI

2.1. Brute force algoritam

Brute force (BF) je metoda gde se koristi „gruba sila" da bi se dobilo jednostavno rešenje. Ona se sastoji od proveravanja svih pozicija u tekstu: da li se početak obrasca (prvi karakter) nalazi tu ili ne? Algoritam ne zahteva pretprocesiranje, i poredi obrazac sa tekstom s leva na desno [1]. Nakon uspešne provere prvog karaktera, obrazac i tekst se pomeraju za tačno jednu poziciju i opet se vrši provera jednakosti – sada drugih karaktera po redu. U slučaju da se karakteri ne podudaraju, prelazi se na proveru sledećeg karaktera teksta (prvi sledeći karakter u odnosu na prvo slovo prethodno proveravanog teksta) i prvog karaktera obrasca. U slučaju da se karakteri podudaraju nastavlja se sa poređenjem narednih karaktera u obrascu i tekstu i tako redom sve do kraja obrasca. Pomeranje se uvek vrši za tačno jedan karakter, zato što algoritam nema nikakve mehanizme koji bi pomogli da se odredi dužina pomeraja.

Što se tiče efikasnosti, u najgorem slučaju broj poređenja je $O(nm)$, gde su n i m dužine teksta i traženog obrasca. Međutim, ako postoje i u tekstu i u obrascu isti karakteri koji se ponavljaju, tada je složenost algoritma $O(m(n-m+1))$. Na primer, ako je obrazac „aaa“, a tekst „aaaaaaaaaa“, tada su $m = 3$ i $n = 10$, i broj poređenja je 24 [4]. Najbolji slučaj je kada se prvi karakter obrasca uopste ne nalazi u tekstu koji se pretražuje i tada je broj poređenja $\Omega(n)$.

2.2. Rabin-Karp algoritam

Rabin-Karp (RK) algoritam koristi heširanje da pronađe traženi obrazac u tekstu. On pokušava da ubrza testiranje podudaranja obrasca i dela teksta koji se trenutno proverava upotrebom heš funkcije. Heš funkcija je funkcija koja pretvara obrazac ili deo teksta koji se proverava u numeričku vrednost, koja se zove heš vrednost. Algoritam koristi činjenicu da ukoliko su obrazac i deo teksta koji se proverava jednaki, onda je jednaka i njihova heš vrednost. Jedan od potencijalnih problema jeste što postoje jednake heš vrednosti za različite obrasce. To znači da ako su heš vrednosti jednake, ne mora da znači da se i obrazac i tekst podudaraju, već se mora izvršiti dodatno proveravanje - karakter po karakter. Glavna ideja ovog algoritma je efikasno izračunavanje heš vrednosti za delove teksta dužine obrasca i za to se koristi *rolling hash* funkcija.
Rabin-Karp algoritam koristi $O(m)$ vremena za pretprocesiranje, dok je u najgorem slučaju vremenska složenost algoritma $O((n-m+1)m)$ [7].

2.3. Knuth-Morris-Pratt algoritam

Knuth-Morris-Pratt (KMP) algoritam je dosta sličan *brute force* metodi, ali najveća razlika je u tome što se, iterator koji se kreće kroz tekst, nikada ne vraća u nazad. Osnovna ideja algoritma je da se u trenutku kada se otkrije neusaglašenost teksta i obrasca (posle podudaranja nekoliko karaktera), već znamo neke od znakova u tekstu sledećeg prozora (deo teksta za koji se trenutno vrši provera o podudaranju). Ta informacija se upotrebljava da se izbegne provera znakova za koje se već zna da će se ionako podudarati.
Algoritam se sastoji iz dva koraka: pretprocesiranje i samo izvršavanje pretrage teksta. U pretprocesiranju se obrađuje sam obrazac. Uz pomoć njega se konstruiše pomoćni niz veličine m (iste veličine kao i obrazac), koji se koristi za preskakanje znakova koji se podudaraju.
Kada se naiđe na neslaganje teksta sa karakterom obrasca, tabela iz pretprocesiranja određuje sa kojim karakterom obrasca treba nastaviti poređenje, a da pri tom pozicija (pokazivač) u tekstu ostane na istom mestu. Ukoliko se indeks, koji pokazuje na obrazac, postavi na 0, to znači da se prvi karakter obrasca i trenutni karakter teksta ne podudaraju, i u tom slučaju se pokazivač teksta pomera za jedno mesto unapred. Ovo znači da se u svakom prolazu pomera ili pokazivač na tekst ili pokazivač na obrazac.
Što se tiče efikasnosti, algoritam KMP ispituje svaki karakter teksta tačno jednom. Vreme rada funkcije koja obrađuje obrazac i od njega pravi pomoćnu tabelu je $O(m)$, što se određuje metodom amortizovane analize. Slična amortizovana analiza, je korišćena i kod ispitivanja složenosti same pretrage. Vreme pretrage je $O(n)$. Pošto se algoritam sastoji iz dva dela, složenost celokupnog KMP algoritma je $O(n+m)$ [3].

Sam algoritam ni nema neku unapređenu verziju sebe. Način za ubrzavanje vremena izvršavanja algoritma jeste paralelizacija. Koncept paralelizacije je uveden radi poboljšanja performansi algoritma. Koristeći koncept paralelizacije, niz veoma dugačkog teksta je podeljen na delove nezavisno od veličine obrasca. Isti obrazac se izvršava na različitim delovima teksta paralelno, smanjujući vreme izvršavanja algoritma. Ovde se KMP algoritam primenjuje na zasebne delove teksta u paraleli. Glavni problem paralelizacije jeste da ako se podudaranje obrasca i teksta nalazi tačno da delu podele podataka ili tački veze. Za rešavanje ovog problema na svakoj tački povezivanja vrši se dodatna obrada podataka. Ukoliko je dužina obrasca m, uzima se $m-1$ karakter od kraja prvog dela, i $m-1$ karakter od početka drugog dela i spoje u jedan string, koji se naknadno proveravaju uz pomoć KMP algoritma.

2.4. Boyer-Moore algoritam

Za razliku od predhodnih algoritama, Boyer-Moore (BM) sadrži tri pametne ideje: skeniranje s desna na levo, „bad character shift rule“ i „good suffix shift rule“ [4, 5].

Boyer-Moore algoritam proverava obrasce skeniranjem karaktera s desna na levo, a ne s leva na desno kao u predhodnim algoritmima. Ideja *lošeg karaktera* je jednostavna. Znak teksta koji se ne podudara sa trenutnim karakterom obrasca naziva se „Bad Character“. Nakon neusklađenosti imamo dva moguća scenarija. Ukoliko se *loš karakter* nalazi u obrascu, u tom slučaju se „poravnava“ obrazac sa tekstom, tako što se poravnaju pokazivači *lošeg karaktera* i odgovarajućeg karaktera obrasca. Sa druge strane ukoliko se *loš karakter* ne nalazi u obrascu, tada se preskače tekst za celu dužinu obrasca. Kod ideje *dobrog sufiksa* posmatramo delove teksta *t* koji se podudaraju sa delovima obrasca. Imamo tri moguća scenarija. Obrazac može da sarži nekoliko istih pojava *t*. U tom slučaju pokušavamo da pomerimo obrazac tako da bi se pojava *t* poklopila sa tekstom. Drugi slučaj je da pokušamo da nadjemo sufiks od *t* koji će se podudariti sa prefiksom od obrasca. I poslednji slučaj je da nemamo uopšte poklapanja obrasca i dela teksta *t*, i tada samo pomerimo obrazac za celu dužinu unapred.

Boyer-Moore algoritam ima najgori slučaj izvođenja $O(n+m)$, ako se uzorak ne pojavljuje u tekstu. Vremenska složenost u fazi pretprocesiranja je $O(m+\delta)$ (δ je broj različitih znakova koji se mogu pojaviti u tekstu). Ukoliko se uzorak pojavljuje u tekstu, vreme izvršavanja algoritma je $O(nm)$ u najgorem slučaju [2, 6]. Iako je ovo ista složenost kao i vremenska složenost *brute force* algoritma, algoritam Boyer-Moore se u praksi bolje pokazuje zbog izostajanja nepotrebnih poređenja.

2.5. Boyer-Moore-Horspool algoritam

Boyer-Moore-Horspool algoritam uvodi tabelu *lošeg podudaranja*. Kada se znakovi ne podudaraju, pretraga skače na sledeće mesto podudaranja u obrascu pomoću tabele. Realizacija ovakvog rešenja može imati kraće vreme izvršavanja od mnogih drugih algoritama pretraživanja jer ne proverava sve znakove teksta, već preskače tekst uz pomoć tabele. Algoritam je brži što je obrazac duži. U najgorem slučaju, performanse algoritma su $O(nm)$, a u najboljem slučaju vreme je $\Omega(n/m)$ [6].

2.6. Boyer-Moore-Horspool-Sunday algoritam

Boyer-Moore-Horspool-Sunday algoritam proširuje ideju korišćenja *lošeg karaktera*. Prilikom ne podudaranja, proverava se sledeći karakter teksta da bi se odredio pomak. Ukoliko se taj karakter ne podudara ni sa jednim karakterom u obrascu, preskače se tekst za dužinu obrasca +1, u suprotnom se proveravani sledeći karakter poravnava sa prvim desnim identičnim karakterom obrasca. Najgora vremenska složenost je $O(nm)$, dok je najbolja vremenska složenost $\Omega(n/m+1)$. Algoritam je brži pri kraćim obrascima [6].

3. TESTIRANJE ALGORITAMA

Algoritmi BF, KMP, RK i BM su napisani u C# programskom jeziku, a zatim su izmerena vremena njihovog izvršavanja nad testnim podacima. Dobijeni rezultati su ovde prikazani i diskutovani.

U eksperimentu je upotrebljen PC računar sa Intel i3-4160 CPU na 3.60GHz i 32GB RAM-a. Operativni sistem je Windows 10 Enterprise 64-bitni.

Testiranje je izvršeno nad dve vrste podataka. Jedna vrsta podataka su veštački generisani primeri koji treba da istaknu dobre i loše strane algoritama, dok drugu vrstu podataka čine realni primeri: tekst knjige od 3226571 karaktera i CIM/XML fajl sa 2GB podataka.

U tabeli 1, prikazano je vreme izvršavanja algoritama nad veštački generisanim primerima, kao i pri različitim dužinama obrasca. Tekstualni fajl nad kojim su izvršena testiranje je bio veličine 274 MB (tj. sadrži 280302624 karaktera).

Tabela 1. *Vremena izvršavanja algoritama nad veštački generisanim primerima*

Obrazac	*m*	BF [s]	KMP [s]	RK [s]	BM [s]	Broj pogodaka
aaaaa aaaab	10	15.75	4.80	6.70	10.6	6673872
aaaaa aaaac	10	15.78	5.55	6.47	9.89	0
aaaaa... ...aaaab	42	37.94	4.38	8.14	9.12	6673872
aaaaa... ...aaaac	42	38.12	7.40	7.57	1.84	0

Kao što se može videti iz ovog primera, BF algoritam ima duže vreme izvršavanja kako se povećava dužina obrasca i gotovo da nije bitno da li ima ili nema pogodaka.

Što se tiče KMP algoritma, najveća razlika se vidi kada ima i kada nema podudaranja. Razlog zašto je algoritam sporiji kad nema podudaranja je taj što algoritam koristi pomoćnu tabelu koja govori sa kojim karakterom obrasca bi trebalo izvršiti poređenje, i tako se pojavljuju dodatna poređenja koja povećavaju vreme izvršavanja.

Na ovom primeru, RK algoritam ima približno isto vreme izvršavanja kada ima i kada nema podudaranja. Takođe, ima malo duže vreme izvršavanja za duže obrasce.

Sa druge strane na BM algoritam znatno utiče veličina obrasca jer algoritam ima mogućnost da u određenim slučajevima preskoči deo teksta za celu dužinu obrasca, što se i desilo u ovom primeru, i iz toga se vidi koliko BM algoritam može biti brži od ostalih u takvim slučajevima.

U drugom testu su algoritmi testirani nad „pravim“ tekstom. U pitanju je knjiga Lava Tolstoja, Rat i mir. Rezultati drugog testa su prikazani u tabeli broj 2.

Tabela 2. *Vremena izvršavanja algoritama nad knjigom Rat i mir*

Obrazac	*m*	BF [s]	KMP [s]	RK [s]	BM [s]	Broj pogodaka
a	1	0.08	0.08	0.10	0.16	195215
been	4	0.09	0.07	0.09	0.07	1476
suddenly	8	0.10	0.08	0.09	0.05	432
illustrious	11	0.09	0.11	0.11	0.05	2

Iz predhodnog primera, pri pretraživanju stvarnog teksta, možemo videti da su rezultati približno jednaki za različite dužine obrazaca. Ukoliko je obrazac dužine jedan karakter, tada je BF metod najbrži, a BM najsporiji, iz razloga što je preskakanje teksta minimalno, što je glavna prednost BM algoritma, i dodatne provere za potencijalno preskakanje su usporile algoritam u odnosu na BF.

Kod najdužeg obrasca BM algoritam ima najbolje vreme izvršavanja, dok RK algoritam ima najsporije vreme izvršavanja.

U trećem testu su algoritmi testirani nad CIM/XML fajlom veličine 2GB. CIM/XML fajl služi za skladištenje, obradu i razmenu podataka. Podaci se sastoje od naziva elementa, koji se nalaze u tagovima, i sadržaja koji daje određenu vrednost elementu i nalazi se između tagova. Elementi, takođe, mogu imati i atribute, koji ih dodatno definišu. Rezultati pretrage nad CIM/XML fajlom su prikazani u tabeli 3.

Tabela 3. *Vreme izvršavanja algoritama nad cim/xml fajlom*

Obrazac	*m*	BF [s]	KMP [s]	RK [s]	BM [s]	Broj pogodaka
agms	4	14.5	19.1	25.5	16.3	19836585
altitude	8	13.4	19.3	24.9	9.3	205014
_100O4L7KZ5X	12	12.6	19.9	25.4	7.3	9
PositionPoint.x Position	23	13.7	21.7	27.1	11.6	4400534

U primeru sa pretragom CIM/XML fajla, za kraće obrasce, dužine do 4 karaktera, najbolje vreme ima *brute force* algoritam, dok najgore vreme ima Rabin-Karp algoritam, iz razloga što je fajl prevelik, i ima dosta pojmova, što znači da algoritam mora da za svaki računa heš funkciju što usporava vreme izvršavanja algoritma.

Prilikom većeg broja karaktera obrasca, Boyer-Moore daje znatno bolje vreme izvršavanja, naročito u slučaju malog broja podudaranja, iz razloga manjeg broja provera u samom algoritmu.

Knuth-Morris-Pratt i Rabin-Karp daju približno isto vreme izvršavanja i kod pretrage realnog teksta, a i kod pretrage CIM/XML fajla.

4. ZAKLJUČAK

U ovom radu su predstavljeni algoritmi *Brute force*, Knuth-Morris-Pratt, Rabin-Karp i Boyer-Moore za pretragu podudaranja obrazaca. Opisani su i varijacije Boyer-Moore algoritma, Boyer-Moore-Horspool i Boyer-Moore-Horspool-Sunday algoritmi. Za svaki od algoritama je opisan način rada i date su složenosti algoritama za najbolje i najgore slučajeve.

Algoritmi su implementirani u C# programskom jezijku i izvršeno je poređenje performansi nad testnim skupovima podataka. Prvo je testirano nad veštački generisanim podacima, dok je sa druge strane testirano nad realnim primerima u vidu pretrage teksta iz knjige i pretrage CIM/XML fajla.

U osnovi, svaki od ovih algoritama donosi dobre rezultate ukoliko je tekst kratak, ali ukoliko se testira nad velikom količinom podataka, i velikim obrascem, vidi se najveća razlika algoritama. Dolazi se do zaključka da ukoliko je obrazac kratak, oko 4 karaktera dužine, najbolje vreme izvršavanja daje Brute force metoda, dok ukoliko je obrazac preko 8 karaktera dužine, najbolje vreme izvršavanja ima Boyer-Moore algoritam.

Najsporije vreme izvršavanja u skoro svim segmentima ima Rabin-Karp algoritam. Knuth-Morris-Pratt algoritam ima približno isto vreme u praksi za različite dužine obrasca.

5. LITERATURA

[1] Herbert S. Wilf, "Algorithms and Complexity", Summer 1994.

[2] C. Charras, T. Lecroq, „Handbook of Exact String-Matching Algorithms", 2004.

[3] http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap34.htm (pristupljeno u septembru 2019.)

[4] D. Gusfield, "Algorithms on Strings, Trees and Sequences. *Computer science and computational biology"*, 1997.

[5] https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/ (pristupljeno u septembru 2019.)

[6] Ramshankar Choudhary, „Variation of Boyer-Moore String Matching Algorithm: A Comparative Analysis", February 2012.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, „Introduction to algorithms", 2001.

Kratka biografija:

Branislav Trkulja rođen je u Novom Sadu 1995. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Primenjeno softversko inženjerstvo odbranio je 2019.god. kontakt: branislavtrkulja@hotmail.com