SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Navarrov algoritam za približno uspoređivanje teksta

Jana Juroš i Josip Skender
Voditelj: *Krešimir Križanović*

Zagreb, svibanj 2021.

SADRŽAJ

1. Uvod

Približno uspoređivanje teksta predstavlja problem pronalaska sekvence koja sadrži neki traženi uzorak, uzimajući u obzir *Levenshteinovu udaljenost uređivanja*. Postoji mnogo algoritama koji obrađuju navedenu problematiku, jedan od njih je *Navarrov algoritam*. U ovom radu opisali smo njegovu funkcionalnost te ga na koncu usporedili s *bit-parallel sequence-to-graph alignment* algoritmom.

2. Navarrov algoritam

Navarrov algoritam oslanja se na *hypertextu* kao strukturi po kojoj pretražujemo najmanju *udaljenost* nekog uzorka.
Najmanja *udaljenost* predstavlja najmanji broj operacija potreban da se neka dva uzorka izjednače. Moguće operacije su: *dodavanje, brisanje i substitucija*.
Hypertext u obliku usmjerenog grafa pohranjuje sadržaj kao čvorove koji u sebi nose *char* informaciju. Svaki čvor sadrži referencu na svoje prethodne i sljedeće čvorove.
Neka je p urozak koji tražimo i G graf po kojem se vrši pretraga. Za provedbu algoritma potrebno je izvršiti i iteracija po svakom čvoru od G, gdje je i duljina od p. Pri iteriranju svaki čvor N poprima vrijednost C koja je jednaka najmanjoj $udaljenosti$ bilokojeg puta u G koji završava u N. Efektivno za određivanje čvorova s najmanjom C vrijednošću za p kroz cijeli algoritam u bilokojem koraku potrebno je samo imati informaciju od prethodne iteracije te pomoću nje izračunavamo trenutnu vrijednost.
U slučaju grafova koji ne sadrže cikluse, obilazak čvorova po redoslijedu ne predstavlja nikakav problem. Kada slijedno obrađujemo čvorove grafa može se desiti da krivo izračunamo C vrijednost čvora u kojem započinje neki ciklus. Ukoliko bi taj čvor imao manju C vrijednost kada bi se uzele u obzir vrijednosti s kraja ciklusa (kasnije vrijednosti po topološkom poretku) dolazi do krivog rješenja.
Da bi se spriječila ova pojava, uklanjamo mogućnost *dodavanja* kod izračuna C te stvaramo novu rekurzivnu funkciju koja za svaki čvor kojem je potrebna redukcija provodi navedenu.

i $\Rightarrow$ *broj iteracije*

Nu $\Rightarrow$ *Skup svih čvorova koji ulaze u N*

E $\Rightarrow$ *skup svih rubova grafa* G

Funkcija za određivanje C vrijednosti:

C = f(N,i) =

  if (p[i] = N.value):

    return min(C $\in$ Nu, i-1)

  else:

    return 1 + min(C $\in$ Nu, C vrijednost od N u prethodnoj iteraciji)

Rekurzivna funkcija Redukcije:

Reduciraj ($u \in Nu$, N):

  if C_N>$1 + C_u$:

    $C_N = 1 + C_u$

    for all x/(N, x $\in$svi sljedbenici od N):

    Reduciraj(x, N)

Gonzalo Navarro algoritam:

GonzaloNavarro (G, p):

  for all $N \in G \leftarrow C_N = 0$

  for $i = 1$ to $m = len(p)$

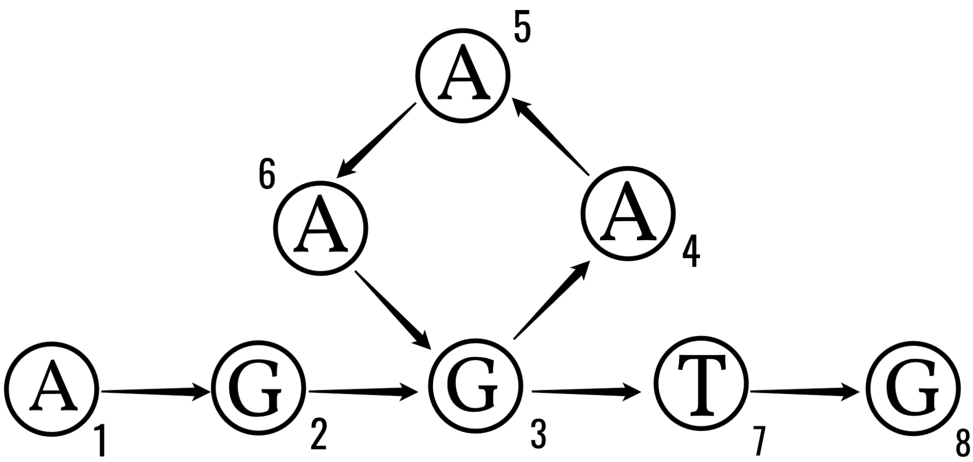
    for all $N \in G$, $C'_n \leftarrow f(N, i)$

    for all $N \in G$, $C_n \leftarrow C'_n$

    for all $(N_u, N) \in G$, Reduciraj(N_u, N)

Primjer funkcionalnosti algoritma:

Usmjereni graf G:



Uzorak p = '**AAAA**'

		1	2	3	4	5	6	7	8
-	C'	0	0	0	0	0	0	0	0
	C	0	0	0	0	0	0	0	0
A	C'	0	1	1	0	0	0	1	1
	C	0	1	1	0	0	0	1	1
A	C'	1	1	1	1	0	0	2	2
	C	1	1	1	1	0	0	2	2
A	C'	2	2	1	1	1	0	2	3
	C	2	2	1	1	1	0	2	3
A	C'	3	3	1	1	1	1	2	3
	C	3	3	1	1	1	1	2	3

Navedena tablica prikazuje svaki korak iteracije algoritma.

Stupci tablice prikazuju vrijednosti C' i C svakog čvora grafa.

C' vrijednost izračuna se po redu za svaki čvor grafa te se tek na kraju iteracije uvrsti u C.

Nakon izračunate C vrijednosti za svaki čvor vrši se provjera **Redukcije** te njena daljnja propagacija.

Tijek izvođenja vrši se po retcima tablice.

C vrijednost u bilokojem koraku prikazuje nam najmanju *Udaljenost* nekog puta koja završava u čvoru s tom vrijednošću.

Ako gledamo 3. redak ispunjene tablice, možemo uočiti da najmanju vrijednost $C = 0$ poprima čvor 6. što nam znači da uspoređivanje uzorka **'AAA'** s G najbolje pristaje

nekom putu koji završava u čvoru 6, u ovom slučaju to bi bio put **4** $\to$ **5** $\to$ **6**. Čije vrijednosti čvorova su upravo jednake '**AAA**'.
Uzmimo sad cijeli p, promatranjem 4. retka tablice uočavamo da je najmanja vrijednost $C = 1$. Nju poprimaju čvorovi **3, 4, 5 i 6**. Ti čvorovi su dio ciklusa grafa G, ovisno koji čvor odaberemo kao kraj puta, vrijednosti čvorova iznose: **AAAG**, **AAGA**, **AGAA** i **GAAA**. Odavde uočavamo da je za svaki od ovih puteva potrebna samo jedna operacije substitucije $G \to A$ i time potvrđujemo izračunatu C vrijednost.

Vremenska složenost ovog algoritma je $O(m(n + e))$.
Prostorna složenost s obzirom da je potrebno znati samo prethodnu iteraciju jednaka je $O(n)$.
$m \to$ duljina od p
$n \to$ broj svih čvorova G
$e \to$ broj svih rubova E.

Funkcionalnost algoritma ovisna je o topološkom poretku grafa G.
Za pronalazak topološkog poretka acikličkih grafova implementirali smo

Khanov algoritam:

L $\leftarrow$ []
S $\leftarrow$ set svih čvorova za koje ne postoji N_u

while S:
    n = S.pop()
    L.append(n)
    for node m $\in$ sljedbenici od n:
        remove E(n,m)
        if m has no more incoming edges:
            S.add(m)

if G has E then:
    return error (*graf sadrži ciklus*)
else:
    return L (*topološki sortiran graf)*

Ako *Khanov algoritam* ustanovi da je graf acikličan tada ne moramo tražiti redukcije i njene propagacije jer smo sigurni da je nemoguće da se takva greška pojavi.
U suprotnom slučaju, za ciklički graf, sortiramo graf topološki jednostavnim DFS algoritmom s dodatnim stogom koji pamti već posjećena stanja te provodimo cijeli Navarrov algoritam s propagacijom redukcije.

3. Usporedba s bit-parallel algoritmom

Za razliku od *Navarrovog algortima*, *bit-parallel algoritam* provodi se u smjeru stupaca gdje su stupci čvorovi grafa a retci vrijednost uzoraka koji uspoređujemo. Svaki red matrice u *Bit-parallel algoritmu* sastoji se od dvaju **bitvectora** (*pozitivan bitvectorP i negativan bitvectorN*) i S_{end} varijable koja čuva rezultat.

Usporedba rezultata naše implementacije **Navarrovog algoritma** i izvornog **Bitvector algoritma**:

Linear graph:

Score:	$77/77$
Bit-parallel time:	3.18 **s**
Navarro time:	57.56 **s**
Time ratio:	18.11
Bit-parallel memory usage:	1844.98KB
Navarro memory usage:	68590.52KB
Memory usage ratio:	36.64

SNP graph:

Score:	$77/77$
Bit-parallel time:	6.06 **s**
Navarro time:	137.95 **s**
Time ratio:	22.76
Bit-parallel memory usage:	4191.48KB
Navarro memory usage:	140389.78KB
Memory usage ratio:	33.50

Tangle graph:

Score:	$77/77$
Bit-parallel time:	32.87 **s**
Navarro time:	99.20 **s**
Time ratio:	3.02
Bit-parallel memory usage:	1992.72KB
Navarro memory usage:	60230.04KB
Memory usage ratio:	30.24

Twopath graph:

Score:	$77/77$
Bit-parallel time:	52.63 **s**
Navarro time:	996.1 **s**
Time ratio:	18.93
Bit-parallel memory usage:	18413.13KB
Navarro memory usage:	136660.70KB
Memory usage ratio:	7.42

4. Zaključak

Navarrov algoritam u konačnosti pronađe ispravno poravnanje za svaki od ispitanih grafova. Uspoređujući **Navarrov algoritam** i **bit-parallel algoritam** jasno je vidljv njegov nedostatak u brzini i uporabi memorije. S obzirom na ulazni skup podataka *Navarrov algoritam* je u prosjeku **13.6** puta sporiji od *bit-parallel algoritma.*

Možemo vidjeti da je naš algoritam očekivano sporiji u odnosu na bit parallel algoritam. Međutim, nismo očekivali ovakvu razliku u uporabi memorije.
Takva razlika je prisutna zato što se u implementaciji *bit parallel* algoritma koristi graf koji sekvencu pohranjuje na poseban, optimalan način.
Naime, tamo je sekvenca umjesto kao jedno polje znakova (*charova*) pohranjena kao dva polja *booleana*.
Budući da imamo četiri moguća znaka - **A, C, G, T** - svaki znak možemo kodirati pomoću dvije boolean vrijednosti (npr. *A →(true, true), C→(true, false), G→(false, true), T→(false, false)*). *Char* je u Javi veličine dva bajta, dok je *boolean* veličine jednog bita. Ako imamo dva polja *booleana* umjesto jednog polja *chara* lako je vidjeti da veličinu grafa možemo svesti na otprilike jednu osminu trenutne veličine grafa.

5. Literatura

[1] Gonzalo Navarro, Improved approximate pattern matching on hypertext, Theoretical Computer Science 237, 2000, pp. 455-463
[2] M.Rautiainen, V.Mäkinen, T.Marschall, Bit-parallel sequence-to-graph alignment, Bioinformatics, Volume 35, Issue 19, 2019, pp. 3599-3607
[3] Topological sorting - Wikipedia, 2021