SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKTNA DOKUMENTACIJA

**LCSk++: mjera sličnosti za dugačke nizove znakova**

Dario Bošnjak

Renato Bošnjak

Dorian Ljubenko

Matija Marić

Matea Torbarina

Zagreb, siječanj, 2019.

**Sadržaj**

[1. Uvod 3](#_Toc535260242)

[2. Opis algoritma 4](#_Toc535260243)

[2.1 Strukture podataka 4](#_Toc535260244)

[2.1.1 Fenwickovo stablo (binarno indeksirano stablo) 4](#_Toc535260245)

[2.2 Način rada algoritma 5](#_Toc535260246)

[2.2.1 Podudarajući par (engl. match pair) 5](#_Toc535260247)

[2.2.2 Algoritam 6](#_Toc535260248)

[2.2.3 Ilustracija rada 7](#_Toc535260249)

[3. Rezultati testiranja 8](#_Toc535260250)

[4. Zaključak 9](#_Toc535260251)

[5. Sažetak 10](#_Toc535260252)

[6. Literatura 11](#_Toc535260253)

# Uvod

Usporedba sličnosti znakovnih nizova bitan je dio mnogih algoritama. Neke od primjena uključuju otkrivanje plagijata, sustave za verzioniranje koda, programe za usporedbu datoteka (alat diff na Unix sustavima) te bioinformatiku.

Često korištene metrike za ovaj problem su najduži zajednički podniz (engl. *longest common subsequence*, LCS) ili dužina uređivanja (engl. *edit distance*).

Zbog vrlo dugačkih nizova koji se pojavljuju kod primjena u bioinformatici (npr. genom), navedene metrike nisu primjenjive.

Metriku LCS za dva znakovna niza moguće je aproksimirati algoritmima (Baker, i dr., 2002) koji imaju vremensku složenost , odnosno , gdje *T* označava broj podudarajućih dijelova u nizovima, a *n, m* duljinu dvaju nizova.

Novi radovi iznijeli su poboljšanja, tako su (Benson, i dr., 2013) definirali mjeru LCS*k* koja pronalazi najveći broj nepreklapajućih podnizova duljine *k*. Predloženi algoritam imao je vremensku i prostornu složenost.

Korištenjem strukture podataka crveno-crnog stabla, poboljšanje predloženog algoritma (Deorowicz, i dr., 2014) donijelo je vremensku složenost koja iznosi i prostornu složenost. Oznake i jednake su kao i u prethodnom algoritmu, oznaka *r* definira ukupni broj podudarajućih podnizova duljine *k* u dva znakovna niza, a oznaka *l* označava duljinu optimalnog rješenja.

Iako su vremenska i prostorna složenost umanjene, algoritam LCS*k* i dalje nije savršen jer pretpostavlja nepreklapajuće podudarajuće podnizove duljine *k*.  
Problem se jasno uočava na sljedećem primjeru: *X* = ABCDE, *Y* = ABCDE, *Z* = ABCFG, *k* = 3. Označimo nepreklapajuća podudaranja podnizova duljine 3 podebljanim slovima:

1. LCS3(X = **ABC**DE, Y = **ABC**DE) = 1
2. LCS3(X = **ABC**DE, Z = **ABC**FG) = 1

Vidimo da LCS3 mjera, za navedene parove ulaznih nizova, iznosi 1 jer se podudarajući podnizovi duljine 3 ne smiju preklapati. Npr. za par ulaza X i Y LCS3 ne uzima u obzir podudaranje podniza CDE jer se preklapa s ABC u znaku C.

Ovaj problem rješava mjera LCS*k*++ (Pavetić, i dr., 2014) koja relaksira uvjet dužine podudarajućih podnizova tako što zahtijeva da oni budu barem duljine *k* čime se izbjegava uvjet nepreklapajućih podnizova.

# Opis algoritma

U ovom poglavlju bit će opisane potrebne strukture podataka i logika algoritma.

## Strukture podataka

Jedna od prednosti mjere LCS*k*++ je jednostavnost implementacije. Algoritam za izračun mjere, u jednostavnoj, ali neefikasnoj varijanti, ne mora koristiti posebne strukture podataka, osim običnog polja.

### Fenwickovo stablo (binarno indeksirano stablo)

Zbog vrlo dugačkih nizova, kakvi se u području bioinformatike često pojavljuju, jednostavna implementacija algoritma postaje beskorisna. Zato se umjesto polja, u jednom dijelu algoritma koristi Fenwickovo stablo, poznato i pod nazivom binarno indeksirano stablo.

#### Problem

Imamo polje A veličine *n* elemenata, želimo ostvariti sljedeće dvije operacije:

1. Povećaj element na *i*-tom mjestu u polju – operacija *update*
2. Izračunaj sumu prvih *i* brojeva u polju A – operacija *query*

Ako koristimo strukturu polja, prvu operaciju možemo izvesti s vremenskom složenosti , dok nam za drugu vremenska složenost u najgorem slučaju iznosi .

#### Rješenje

Korištenje Fenwickovog stabla omogućuje izvođenje obje operacije s vremenskom složenosti , dok se memorijska složenost ne mijenja u odnosu na strukturu polja. Umjesto operacije dohvaćanja sume prvih *i* brojeva, algoritam LCS*k*++ koristi modificirano Fenwickovo stablo za sljedeće operacije, bez promjene vremenske složenosti:

1. Postavi element na *i*-tom mjestu u polju na vrijednost *v*, ako vrijedi *v* > A[*i*] – operacija *update*
2. Dohvati najveći od prvih *i* elementa polja – operacija *max*

## Način rada algoritma

### Podudarajući par (engl. match pair)

Algoritam radi s parovima indeksa koje zovemo podudarajući parovi (engl. *match pairs)*.

Za nizove X i Y definiramo:

Par indeksa je podudarajući par ako je , tj. ako je podniz niza X koji počinje na poziciji te je duljine jednak podnizu niza Y koji počinje na poziciji te je duljine .

Za svaki podudarajući par u tablici dinamičkog programiranja pamtimo vrijednost .

Zbog mogućih dugačkih nizova, a radi uštede memorije, tablicu dinamičkog programiranja poželjno je ostvariti pomoću tablice raspršenog adresiranja u kojoj su parovi ključevi.

#### Prethođenje podudarajućih parova

Podudarajući par prethodi podudarajućem paru ako vrijedi:

.

#### Nastavljanje podudarajućih parova

Podudarajući par nastavlja podudarajući par ako vrijedi:

.

Tablica 2.1prikaz tablice dinamičkog programiranja i nastavljanja podudarajućih parova za k=3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| G |  |  |  |  |  |
|  | P |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  | G |  |  |
|  |  |  |  |  |  |

U tablici su prikazani podudarajući parovi P i G koji su u relaciji P nastavlja G.

### Algoritam

#### Ulazi algoritma

Ulazi u algoritam su niz X duljine te niz Y duljine te prirodni broj .

#### Objašnjenje rada algoritma

Algoritam za dva niza X i Y u početku odredi sve parove indeksa za koje je . Svaki podudarajući par je duljine i ima svoj početak i kraj iz čega vidimo da su krajnji indeksi isključujući. Početke i krajeve parova nazivamo događajima (engl. *events*). Događaji se izdvoje u posebnu listu te se uzlazno sortiraju po prvom pa drugom indeksu u paru. (engl. *row-major order*). Ako postoje dva događaja s istim indeksima događaj kraja ima prednost.

Uz listu događaja treba nam i tablica dinamičkog programiranja (DP) veličine za pamćenje međurezultata te binarno indeksirano stablo (BIT) veličine u koje pamtimo najveći dosadašnji rezultat po stupcima tablice dinamičkog programiranja.

Zatim se radi iteriranje po svim događajima u kojem se razmatraju dva slučaja.

U slučaju događaja početka tablicu dinamičkog programiranja za taj događaj ažuriramo na vrijednost . Objašnjenje je sljedeće. Kako događaj početka označava početne indekse podnizova u oba niza koji su jednaki, rješenje treba uvećati za . Dodatno rješenje treba uvećati za najveće dosadašnje rješenje po stupcima tablice dinamičkog programiranja (osjenčani dio tablice).

Tablica 2.2 ažuriranje tablice dinamičkog programiranja u slučaju događaja početka

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | x | x | x | x | x |
| x | x | x | x | x | x |
| x | x | x | o |  |  |
|  |  |  |  |  |  |

Znakovi **x** u tablici označavaju dosadašnja rješenja, dok znak **o** označava trenutno promatrani događaj početka. U praznim poljima zapisane su vrijednosti 0 pa ona ne utječu na pronalazak najveće vrijednosti po stupcima.

U slučaju događaja kraja (označimo ga s ) najprije tražimo početni događaj G takav da nastavlja G te, ako takav G postoji, ažuriramo tablicu dinamičkog programiranja na vrijednost što slijedi iz definicije nastavljanja podudarajućih parova.  
Zatim se binarno indeksirano stablo uvjetno ažurira s kako bi zapamtili novo najbolje rješenje.

#### Izlaz algoritma

Algoritam kao izlaz daje najveću vrijednost iz tablice dinamičkog programiranja koja predstavlja broj sličnost nizova X i Y.

### Ilustracija rada

X = ABCDE  
Y = ABCDE  
Z = ABCFG  
k = 3

U uvodu smo vidjeli da klasični algoritam LCS*k* za nizove X i Y daje rješenje 1, kao i za nizove X i Z koji su međusobno manje slični.

Algoritam LCS*k++* za par nizova X i Y daje rješenje 5, dok za manje sličan par nizova X i Z daje rješenje 3. Vidimo da ova mjera sličnosti bolje odražava stvarnu sličnost nizova.

# Rezultati testiranja

Tablica 3.1 Rezultati implementacija u različitim programskim jezicima

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **C++** | | **Python** | | **Scala** | | **Java** | | **TypeScript** | |
|  | Duljina sekvence | *k* | **Memorija [kB]** | **Vrijeme izvođenja [ms]** | **Memorija [kB]** | **Vrijeme izvođenja [ms]** | **Memorija [kB]** | **Vrijeme izvođenja [ms]** | **Memorija [kB]** | **Vrijeme izvođenja [ms]** | **Memorija [kB]** | **Vrijeme izvođenja [ms]** |
| Sintetski podaci | 1,00E+02 | 10 | 62 | 12 | 12496 | 16 | 2215 | 62 | 498 | 78 | 6666 | 0.5 |
| 20 | 62 | 7 | 12460 | 15 | 2203 | 46 | 498 | 78 | 6666 | 0.5 |
| 30 | 62 | 15 | 12500 | 8 | 2206 | 47 | 498 | 78 | 6658 | 0.5 |
| 1,00E+04 | 10 | 5900 | 1386 | 12713 | 548 | 8855 | 124 | 593 | 113 | 10419 | 17 |
| 20 | 7800 | 1736 | 12485 | 593 | 8977 | 108 | 577 | 114 | 11591 | 14 |
| 30 | 7800 | 1777 | 12517 | 621 | 9830 | 97 | 577 | 106 | 11596 | 15 |
| 1,00E+06 | 10 | 329400 | 84209 | 233099 | 7613104 | 313942 | 12665 | 189318 | 8169 | 446246 | 12305 |
| 20 | 650000 | 54273 | 9564 | 10012421 | 249598 | 3907 | 8311 | 2296 | 312753 | 2038 |
| 30 | 649900 | 153156 | 9490 | 5930488 | 258088 | 4010 | 8312 | 2249 | 304343 | 2022 |
| E. coli | 1,00E+06 | 10 | 565600 | 272523 | 467005 | 4691401 | 856842 | 32416 | 734581 | 35200 | 1223461 | 74215 |
| 20 | 740600 | 163160 | 21778 | 5359963 | 418701 | 12977 | 317618 | 10659 | 1086852 | 16236 |
| 30 | 709900 | 141420 | 27734 | 5449237 | 367407 | 11843 | 259585 | 10143 | 484794 | 13050 |

U prikazanoj tablici mogu se vidjeti neke od performansi implementacija algoritma u 5 različitih jezika. Moguće je uočiti kako se vrijeme izvršavanja programa i zauzeće memorije povećavaju prilikom testiranja većih sekvenci. Za veće vrijednosti parametra *k* zauzeće memorije i vrijeme izvršavanja programa se smanjuju.

# Zaključak

U sklopu projekta usvojeni su principi i implementiran je LCSk++ algoritam. Svrha LCSk++ algoritama je pronalazak najdužeg zajedničkog podniza dvaju nizova.

Strukturu za pronalazak najvećeg elementa u prvih *j* elemenata, radi vremenske učinkovitosti, potrebno je implementirati kao Fenwickovo stablo, a za čuvanje vrijednosti za svaki par poželjno je koristiti mapu raspršenog adresiranja (engl. *hash map*). Iako implementacija može uspješno obrađivati nizove proizvoljnih tipova podataka, naš cilj bila je obrada bioloških nizova genoma u FASTA formatu. Ovaj algoritam pogodan je za analizu niza podataka reda veličine 106 znakova, koji je ujedno red veličine sekvence bakterije *Escherichia Coli*, na kojoj je algoritam testiran.

# Sažetak

LCSk++ predstavlja nadograđenu verziju LCS algoritama za pronalaženje najdužeg zajedničkog podniza dvaju nizova. U odnosu na LCSk i ranije verzije algoritma, koji pronalaze isključivo nepreklapajuće zajedničke podnizove, LCSk++ omogućuje produživanje zajedničkog podniza na *k* ili više znakova i zbog toga daje najbolje rezultate. U nalaženju zajedničkih podnizova primijenjen je princip dinamičkog programiranja, a radi poboljšanja brzine i složenosti koriste se strukture kao što su Fenwick *tree* i *hash* mapa. U konačnici se dobivaju linearna vremenska i memorijska složenost. U sklopu ovog projekta, algoritam je implementiran u pet različitih programskih jezika i verificirana je točnost pojedinih implementacija. Također, provedeno je usporedno testiranje vremenskih i memorijskih performansi implementacija.

# Literatura

**Baker, Brenda i Giancarlo, Raffaele. 2002.** Sparse dynamic programming for longest common subsequence from fragments. *Journal of Algorithms.* 2002, Svez. 42, 2, str. 231--254.

**Benson, Gary, Levy, Avivit i Shalom, Riva. 2013.** *International Conference on Similarity Search and Applications.* s.l. : Springer, 2013. str. 257--265.

**Deorowicz, Sebastian i Grabowski, Szymon . 2014.** *Information Processing Letters.* s.l. : Elsevier, 2014. str. 634--638. Svez. 114.

**Pavetić, Filip, Žužić, Goran i Šikić, Mile . 2014.** LCSk++: Practical similarity metric for long strings. *arXiv preprint.* 2014.