Longest common subsequence korišćenjem Genetskog algoritma

Projekat iz Računarske inteligencije
Matematički fakultet
Univerzitet u Beogradu

Pavle Cvejović
mi18024@alas.matf.bg.ac.rs
Viktor Novaković
mi18096@alas.matf.bg.ac.rs

September 2022

Apstrakt

Algoritmi na sekvencama[1] simbola su izučavani duže vreme i sada formiraju fundamentalni deo računarskih nauka. Jedan od veoma bitnih problema u analiziranju sekvenci je problem pronalaženja najduže zajedničke podsekvence. U generalnom slučaju, kada imamo proizvoljan broj ulaznih sekvenci, ovaj problem je NP-težak [2]. Ovde opisujemo pristup za rešenje ovog problema baziran na genetskom algoritmu.

[1] Koristimo termin *sekvenca* a ne *niska* jer se ovaj problem odnosi na nizove proizvoljnih tipova i ne koristimo termin *niz* jer će se on upotrebljavati u drugom kontekstu

Sadržaj

1 Uvod

Problem pronalaženja najduže zajedničke podsekvence skupa od k sekvenci (k-Longest Common Subsequence, k-LCS) je jedan od najizučavanijih problema u računarskim naukama u posednjih 30-ak godina jer igra bitnu ulogu u poredjenju sekvenci podataka. Ima potencijalne primene u mnogim područijima. Ovaj problem je koristan u prepoznavanju uzoraka, obradi i kompresiji teksta i podataka [3] i molekularnoj biologiji [1]. Može se posmatrati kao *mera bliskosti* k sekvenci jer se sastoji iz pronalaženja najvećeg broja identičnih elemenata svih k sekvenci ali takvih da je bitno uredjenje tih elemenata. Na primer, poredjenje k DNK sekvenci da se vidi njihovo podudaranje ili traženje reči u rečniku blizu pogrešno napisane reči u aplikacijama koje proveravaju pravopis.

2 Opis problema

Ako imamo dve sekvence S i T na nekoj fiksnoj azbuci Σ, sekvenca T je *podsekvenca* od S ukoliko se T može dobiti iz S brisanjem nekih elemenata iz S. Primetimo da uredjenje preostalih elemenata u S mora biti očuvano. Dužina sekvence S je broj elemenata u njoj i zapisuje se sa $|S|$. Radi jednostavnosti, sa $S[i]$ ćemo obeležavati i-ti element u sekvenci S, a sa $S[i,j]$ podsekvencu od S koja se sastoji od i-tog do j-tog elementa iz S.

Problem: Ako su nam date sekvence $S_i, 1 \leq i \leq k$, na nekoj fiksnoj azbuci Σ, pronadji sekvencu T koja je podsekvenca S_i za svako $i \in \{1, 2, ..., k\}$

3 Brute force algoritam

3.1 Implementacija

Za implementaciju *brute force* algoritma koristili smo tehniku dinamičkog programiranja pronalaženja LCS od k nizova. Prvo, podsetimo se kako izgleda funkcija koja prima kao argumente dužine (n_1, n_2) 2 niza (a_1, a_2) i računa dužinu njihovog LCS:

$$f(n_1, n_2) = \begin{cases} 0 & \text{ako } n_1 = 0 \vee n_2 = 0 \\ f(n_1 - 1, n_2 - 1) + 1 & \text{ako } a_1[n_1 - 1] = a_2[n_2 - 1] \\ \max\left(f(n_1 - 1, n_2), f(n_1, n_2 - 1)\right) & \text{inače} \end{cases}$$

Ukoliko bi želeli da generalizujemo ovu funkciju za k nizova $a_1, a_2, ..., a_k$ sa dužinama $n_1, n_2, ...n_k$ dobili bi smo sledeću rekurzivnu formulu:

$$f(n_1, n_2, ..., n_k) = \begin{cases} 0, & \text{ako je } n_1 = 0 \vee n_2 = 0 \vee \cdots \vee n_k = 0 \\ f(n_1 - 1, n_2 - 1, ..., n_k - 1) + 1, & \text{ako je } a_1[n_1 - 1] = a_2[n_2 - 1] = \cdots = a_k[n_k - 1] \\ \max\left(f(n_1 - 1, n_2, ..., n_k), f(n_1, n_2 - 1, ..., n_k), ..., f(n_1, n_2, ..., n_k - 1)\right), & \text{inače} \end{cases}$$

Dalje, kreirajmo k-dimenzioni niz gde je svaki od nizova iste duzine kao odgovarajuća sekvenca (i inicijalizujmo ih tako da su im svi elementi 0) koji će nam koristiti kao DP tablica za enumeraciju svih sekvenci za računanje puta od LCS. Korišćenjem prethodne formule, možemo jednostavno da izvučemo formulu i za formiranje DP tablice koja će nam pomoći u rekonstrukciji LCS:

Algoritam 1 Formiranje DP tablice

Ulaz: niz sekvenci $a_1, a_2, ..., a_k$ **i njihovih dužina** $n_1, n_2, ...n_k$
Izlaz: formirana k**-dimenziona DP tablica**

$\texttt{DP} \leftarrow \mathbf{0}_{n_1 \times n_2 \times \cdots \times n_k}$
for $(i_1, i_2, ..., i_k), (el_1, el_2, ..., el_k)$ `in` enumerate$(a_1 \times a_2 \times \cdots \times a_k)$ **do**
  if $el_1 = el_2 = \cdots = el_k$ **then**
    $DP[i_1, i_2, ..., i_k] \leftarrow DP[i_1 - 1, i_2 - 1, ..., i_k - 1] + 1$
  else
    $DP[i_1, i_2, ..., i_k] \leftarrow \max(DP[i_1 - 1, i_2, ..., i_k], DP[i_1, i_2 - 1, ..., i_k], ..., DP[i_1, i_2, ..., i_k - 1])$
  end if
end for
return `DP`

Sada možemo ići "unazad" kroz DP tablicu da rekonstruišemo LCS tako što kada god nadjemo pogodak u tablici nadovežemo element iz bilo koje sekvence (koristeći bilo koji indeks) na rezultujuću podniz:

Algoritam 2 Formiranje LCS

Ulaz: k**-dimenziona tablica DP, niz sekvenci** $a_1, a_2, ..., a_k$ **i njihovih dužina** $n_1, n_2, ...n_k$
Izlaz: LCS ulaznih sekvenci

`lcs` $\leftarrow$ `[]`
while $n_i > 0 \ \ (\forall i \in \{1, .., k\})$ **do**
  `step` $\leftarrow DP[n_1, n_2, ..., n_k]$
  if $(\exists i \in \{1, ..., k\})$ `step` $= DP[n_1, n_2, ..., n_i - 1, ..., n_k]$ **then**
    $n_i \leftarrow n_i - 1$
  else
    `lcs.append(`$a_1[n_1 - 1]$`)`
    $n_i \leftarrow n_i - 1 \ \ (\forall i \in \{1, .., k\})$
  end if
end while
return `lcs`

4 Genetski algoritam

4.1 Uopšteno

4.2 Fitnes funkcija

4.3 Implementacija

5 Testiranje i rezultati

5.1 Brute force

5.2 Genetski

6 Zaključak

Literatura

[1] R. Beal et al. “A new algorithm for “the LCS problem” with application in compressing genome resequencing data”. In: (2016), pp. 370–380.

[2] David Maier. “The Complexity of Some Problems on Subsequences and Supersequences”. In: (1978), pp. 322–336.

[3] J. Storer. “Data Compression: Methods and Theory”. In: (1988).