SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Projekt

Bioinformatika

Određivanje LCP polja korištenjem modificiranog algoritma SA-IS

Marko Posavec

Luka Šopar

Mentor: Doc. dr. sc. Mirjana Domazet-Lošo

Zagreb, siječanj 2020.

Sadržaj

[1. Kratki opis problema 4](#_Toc29933448)

[1.1. Zadatak 4](#_Toc29933449)

[2. Modifikacija algoritma 5](#_Toc29933450)

[3. Primjer algoritma 8](#_Toc29933451)

[4. Minimalna vrijednost 11](#_Toc29933452)

[5. Usporedba s originalnom implementacijom. 12](#_Toc29933453)

[6. Upute za instalaciju 14](#_Toc29933454)

[7. Literatura 15](#_Toc29933455)

Popis slika

[Slika 1 SA-IS algoritam [1] 4](#_Toc29853554)

[Slika 2 Induciranje SA više razine [1] 5](#_Toc29853555)

[Slika 3 Podaci za izvedeni primjer[1] 8](#_Toc29853556)

Popis tablica

[Tablica 1 Prvi korak 8](#_Toc29853558)

[Tablica 2 Drugi korak 9](#_Toc29853559)

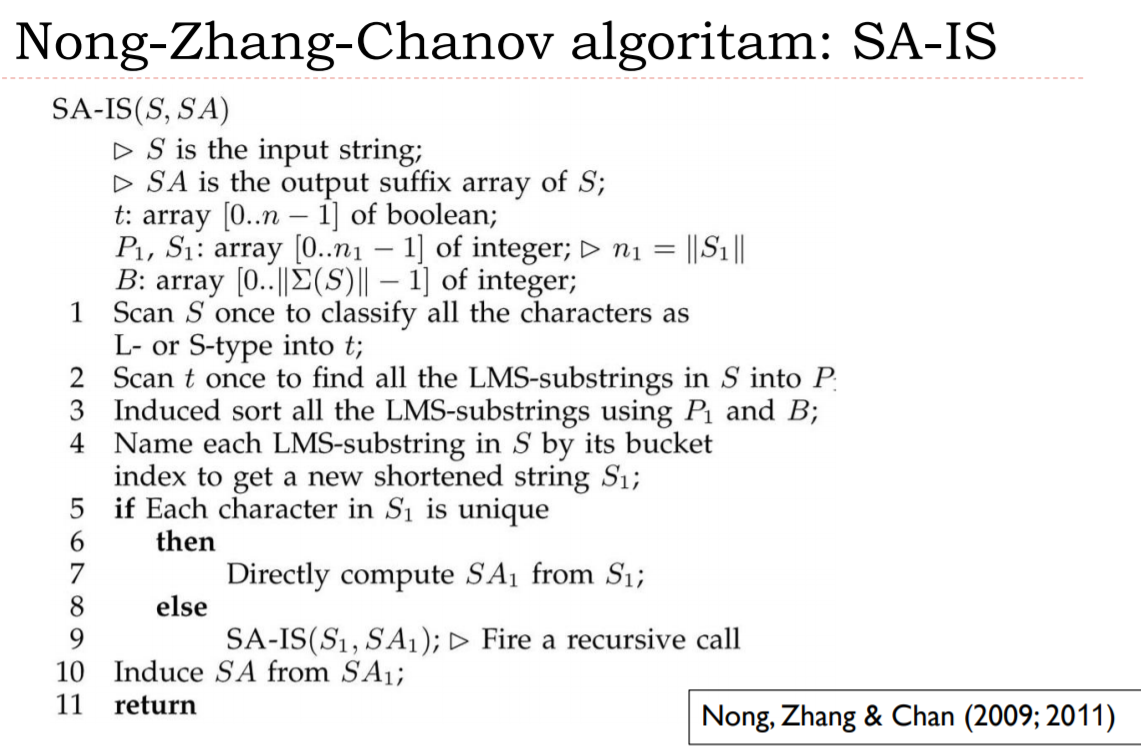
[Tablica 3 Treći korak 10](#_Toc29853560)

# Kratki opis problema

## Zadatak

Zadatak projekta je korištenjem modificiranog SA-IS algoritma odrediti LCP polje.

SA-IS algoritam je algoritam koji računa sufiksno polje (eng. suffix array - SA) pomoću induciranog sortiranja. Sufiksno polje, polje je koje sadrži indekse sortiranih sufiksa iz originalnog polja.



Slika 1 SA-IS algoritam [1]

U projektu je implementiran algoritam prema pseudokodu iz predavanja 5.

Algoritam je potrebno modificirati kako bi se uz sufiksno polje izračunalo i LCP polje [2].

LCP polje (eng. longest common prefix array) je pomoćna struktura podataka sufiksnom polju. Sadrži duljine najduljih zajedničkih prefiksa svih parova sufiksa koji su susjedni unutar sufiksnog polja.

# Modifikacija algoritma

LCP polje također se računa induciranjem. Potrebno je izmijeniti korak u kojem se iz SA polja niže razine računa SA polje više razine (koraci 7 i 10 na slici 1).

Proširenje se može odnositi na sve rekurzivne pozive SA-IS algoritma te se i LCP polje inducira rekurzivno ili se LCP polje može inducirati zajedno sa najvišom razinom sufiksnog polja.

Prikazano proširenje odnosni se na induciranje LCP polja zajedno s najvišom razinom sufiksnog polja.

Slika na kojoj se prikazuje tekst

Opis je automatski generiran

Slika 2 Induciranje SA više razine [1]

Induciranje SA više razine sastoji se od tri koraka. Svaki korak potrebno je proširiti kako bi se induciralo i LCP polje.

Proširenja:

1. korak

* Za svaki P2[SA2[i]] koji se ubacuje od kraja pretinca, naivno izračunati LCP vrijednost tog sufiksa i sufiksa njemu slijeva (direktnom usporedbom u originalnom polju) te LCP vrijednost zapisati na odgovarajuću lokaciju: B1[k] + 1. LCP trenutne pozicije postaviti na 0. LCP[B1[k]] = 0.
* Pomaknuti B1[k] za jedno mjestu ulijevo.

2. korak – zapisuju se L sufiksi

* Ako se L sufiks S[SA[i] - 1] zapisuje na početak odgovarajućeg pretinca, njegova LCP vrijednost je 0. LCP[B1[k]] = 0.
* Ako se L sufiks ne zapisuje na početak odgovarajućeg pretinca, pretpostaviti da mu je lijevi susjed u pretincu izračunat u iteraciji i'.

i' < i.

* + Ako su i te i' u različitim pretincima, znači da sufiksi S[SA[i]] i S[SA[i']] počinju različitim znakovima pa L sufiks koji se zapisuje ima s lijevim susjedom zajednički samo jedan znak. LCP[B1[k]] = 1.
  + Ako su i te i' u istom pretincu potrebno je naći minimum l unutar polja LCP između indeksa (i'+1) te i. l = RMQ(LCP, i'+1, i). Sva mjesta LCP polja između i'+1 te i su u istom pretincu i već su izračunata. LCP[B1[k]] = l + 1.
* Napomena: potrebno je pripaziti da se prilikom upisa zadnjeg L sufiksa u pretinac osvježi LCP vrijednost prvog S sufiksa u tom pretincu.

Primjer: (4, -1, 6, 7). 6 i 7 su S sufiksi, a 4 je posljednje upisani L sufiks. Potrebno je osvježiti LCP za sufiks 6, naivno računajući LCP vrijednost između 4 i 6.

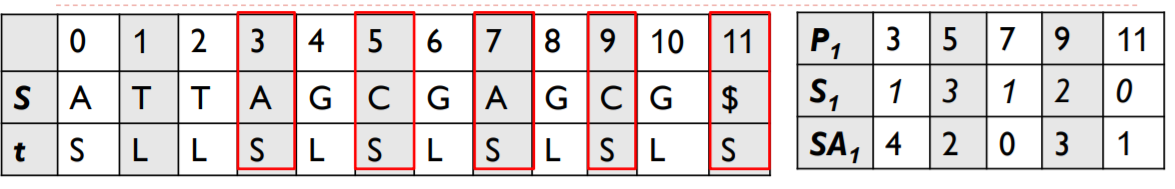
\*RMQ (eng. range minimum query) ovisno o definiciji vraća minimalnu vrijednost ili indeks minimalne vrijednosti nekog polja između dva zadana indeksa. U ovom slučaju, neka vraća minimalnu vrijednost.

3. korak – zapisuju se S sufiksi (simetričan korak prethodnom)

* Ako se S sufiks S[SA[i] - 1] zapisuje na kraj odgovarajućeg pretinca, preskače se induciranje te se vodi računa samo o napomeni
* Ako se S sufiks ne zapisuje na početak odgovarajućeg pretinca, pretpostaviti da mu je desni susjed u pretincu izračunat u iteraciji i'. i' > i.
  + Ako su i te i' u različitim pretincima, znači da sufiksi S[SA[i]] i S[SA[i']] počinju različitim znakovima pa S sufiks koji se zapisuje ima s desnim susjedom zajednički samo jedan znak. LCP[B1[k] + 1] = 1.
  + Ako su i te i' u istom pretincu potrebno je naći minimum l unutar polja LCP između indeksa i te (i'+1). l = RMQ(LCP, i+1, i'). Sva mjesta LCP polja između i te i'+1 su u istom pretincu i već su izračunata. LCP[B1[k] + 1] = l + 1.
* Napomena: potrebno je pripaziti da se prilikom upisa zadnjeg S sufiksa (s desna na lijevo) u pretinac osvježi LCP vrijednost prvog L sufiksa (s desna na lijevo) u tom pretincu.

Primjer: (4, 9, 6, 7). 9, 6 i 7 su S sufiksi, a 4 je L sufiks. Prilikom upisivanja 9 u SA, potrebno je osvježiti LCP za sufiks 9, naivno računajući LCP vrijednost između 4 i 9.

# Primjer algoritma



Slika 3 Podaci za izvedeni primjer[1]

Prikazani zadnji korak – induciranje SA te LCP polja.

1. korak

Tablica 1 Prvi korak

|  |  |  |
| --- | --- | --- |
| i | SA  LCP | Komentar |
| i = 4 | {-1} {-1, -1, -1} {-1, 5} {-1, -1, -1, -1} {-1, -1}  {-1}{-1, -1, -1}{-1, 0}{0, -1, -1, -1}{-1, -1} | Dodaje se na kraj pretinca. Izračuna se LCP sa sljedećim – nema zajedničkih znakova jer su u različitim pretincima |
| i = 3 | {-1} {-1, -1, -1} {9, 5} {-1, -1, -1, -1} {-1, -1}  {-1}{-1, -1, -1}{0, 2}{0, -1, -1, -1}{-1, -1} | Naivno se izračuna se LCP između sufiksa 9 i 5 |
| i = 2 | {-1} {-1, -1, 3} {9, 5} {-1, -1, -1, -1} {-1, -1}  {-1}{-1, -1, 0}{0, 2}{0, -1, -1, -1}{-1, -1} | Dodaje se na kraj pretinca. Izračuna se LCP sa sljedećim – nema zajedničkih znakova jer su u različitim pretincima |
| i = 1 | {-1} {-1, 7, 3} {9, 5} {-1, -1, -1, -1} {-1, -1}  {-1}{-1, 0, 4}{0, 2}{0, -1, -1, -1}{-1, -1} | Naivno se izračuna se LCP između sufiksa 7 i 3 |
| i = 0 | {11} {-1, 7, 3} {9, 5} {-1, -1, -1, -1} {-1, -1}  {0}{0, 0, 4}{0, 2}{0, -1, -1, -1}{-1, -1} | Dodaje se na kraj pretinca. Izračuna se LCP sa sljedećim – nema zajedničkih znakova jer su u različitim pretincima |

1. korak

Tablica 2 Drugi korak

|  |  |  |  |
| --- | --- | --- | --- |
| i | i' | SA LCP | Komentar |
| i = 0 | i' = - | {11} {-1, 7, 3} {9, 5} {10, -1, -1, -1} {-1, -1}  {0}{0, 0, 4}{0, 2}{0, -1, -1, -1}{-1, -1} | Dodaje se na početak pretinca. Potrebno LCP postaviti na 0. |
| i = 2 | i' = 1 | {11} {-1, 7, 3} {9, 5} {10, 6, -1, -1} {-1, -1}  {0}{0, 0, 4}{0, 2}{0, 1, -1, -1}{-1, -1} | i i i' u različitim su pretincima. Potrebno LCP postaviti na 1.  \*provjeriti napomenu: 7 je prvi S sufiks u pretincu, provjeriti postoji li prije njega u pretincu negdje L sufiks i po potrebi osvježiti za S sufiks LCP. |
| i = 3 | i' = - | {11} {-1, 7, 3} {9, 5} {10, 6, -1, -1} {2, -1}  {0}{0, 0, 4}{0, 2}{0, 1, -1, -1}{0, -1} | Dodaje se na početak pretinca. Potrebno LCP postaviti na 0. |
| i = 4 | i' = 2 | {11} {-1, 7, 3} {9, 5} {10, 6, 8, -1} {2, -1}  {0}{0, 0, 4}{0, 2}{0, 1, 1, -1}{0, -1} | i i i' u različitim su pretincima. Potrebno LCP postaviti na 1.  \*provjeriti napomenu: 9 je prvi S sufiks u pretincu, provjeriti postoji li prije njega u pretincu negdje L sufiks i po potrebi osvježiti za S sufiks LCP. |
| i = 5 | i' = 4 | {11} {-1, 7, 3} {9, 5} {10, 6, 8, 4} {2, -1}  {0}{0, 0, 4}{0, 2}{0, 1, 1, 3}{0, -1} | i i i' u istim su pretincima. Potrebno LCP postaviti na minimum u LCP polju u intervalu [5,5].  RMQ(5, 5) = 2 |
| i = 10 | i' = 3 | {11} {-1, 7, 3} {9, 5} {10, 6, 8, 4} {2, 1}  {0}{0, 0, 4}{0, 2}{0, 1, 1, 3}{0, 1} | i i i' u različitim su pretincima. Potrebno LCP postaviti na 1. |

1. korak

Tablica 3 Treći korak

|  |  |  |  |
| --- | --- | --- | --- |
| i | i' | SA  LCP | Komentar |
| i = 11 | i' = - | {11} {-1, 7, 0} {9, 5} {10, 6, 8, 4} {2, 1}  {0}{0, 0, 4}{0, 2}{0, 1, 1, 3}{0, 1} | Dodaje se na kraj pretinca. Nije potrebno ništa inducirati.  \*provjeriti napomenu (je li sufiks lijevo od upisanog L i u istom pretincu. Ako jest, osvježiti LCP na trenutnoj poziciji). Potrebno provjeriti u svakom sljedećem retku. |
| i = 9 | i' = 11 | {11} {-1, 3, 0} {9, 5} {10, 6, 8, 4} {2, 1}  {0}{0, 0, 1}{0, 2}{0, 1, 1, 3}{0, 1} | i i i' su u različitim pretincima. Potrebno postaviti LCP pozicije s desne strane na 1. |
| i = 8 | i' = 9 | {11} {7, 3, 0} {9, 5} {10, 6, 8, 4} {2, 1}  {0}{0, 4, 1}{0, 2}{0, 1, 1, 3}{0, 1} | i i i' su u istom pretincu. Potrebno je LCP s desne strane postaviti na minimum u intervalu [9, 9] uvećan za jedan.  RMQ(9, 9) = 3 |
| i = 7 | i' = - | {11} {7, 3, 0} {9, 5} {10, 6, 8, 4} {2, 1}  {0}{0, 4, 1}{0, 2}{0, 1, 1, 3}{0, 1} | Dodaje se na kraj pretinca. Nije potrebno ništa inducirati. |
| i = 6 | i' = 7 | {11} {7, 3, 0} {9, 5} {10, 6, 8, 4} {2, 1}  {0}{0, 4, 1}{0, 2}{0, 1, 1, 3}{0, 1} | i i i' su u istom pretincu. Potrebno je LCP s desne strane postaviti na minimum u intervalu [7, 7] uvećan za jedan.  RMQ(7, 7) = 3 |

# Minimalna vrijednost

Pronalazak minimalne vrijednosti u intervalu unutar polja je RMQ problem. Tri se parametra uzimaju u obzir od rješavanja problema. Memorijska složenost za pomoćne strukture, vremenska složenost za stvaranje tih struktura te vremenska složenost izračuna minimalne vrijednosti.

Polazni rad [2], definira postupak rješavanja tog problema. Navedeni postupak puno je općenitiji i kompliciraniji nego što je potrebno.

Originalna implementacija [3] koristi MinStack za odgovaranje na RMQ upite. U ovdje opisanoj implementaciji, preuzeto je isto rješenje. Više o RMQ upitima koristeći MinStack, opisano je u literaturi [4].

# Usporedba s originalnom implementacijom.

Za izračun brzine izvođenja algoritama korištena je funkcija clock(), prije i nakon poziva sa\_is rekurzije:

clock\_t start = clock();

sa\_is(2, temp.length(), sa, lcp, 256, t, p1);

clock\_t finish = clock();

cout << "Duration time: " << (double)(finish - start)/(double)CLOCKS\_PER\_SEC;

Memoriju smo mjerili na Windows računalima, koristeći ugrađene mehanizme:

void print\_memory\_usage()

{

//get the handle to this process

auto myHandle = GetCurrentProcess();

//to fill in the process' memory usage details

PROCESS\_MEMORY\_COUNTERS pmc;

//return the usage (bytes), if I may

if (GetProcessMemoryInfo(myHandle, &pmc, sizeof(pmc)))

{

auto size = pmc.PeakWorkingSetSize;

cout <<"Peak set:" << size << endl;

}

else

return;

}

U originalnu implementaciju dodan je isti kod, prilagođen programskom jeziku C. Ispisuje najveću vrijednost WorkingSeta koji je bio korišten za vrijeme izvođenja – veći je nego što je stvarno korištenje programa, no najbolja je mjera za računanje na Windows platformi.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Veličina primjera | Brzina naše  (s) | Brzina originalne  (s) | Korištena memorija naše | Korištena memorija originalne | Točnost (isti rezultat) |
| 25 B | 0 | 0 | 3 207 168 B | 3 588 096 B | Da |
| 50 B | 0 | 0 | 3 575 808 B | 3 227 648 B | Da |
| 200 B | 0 | 0 | 3 588 096 B | 3 182 592 B | Da |
| 1000 B | 0 | 0 | 3 596 288 B | 3 203 072 B | Da |
| 5000 B | 0.001 | 0.001 | 3 702 784 B | 3 272 704 B | Da |
| 50 000 B | 0.008 | 0.009 | 4 530 176 B | 3 670 016 B | Da |
| 100 000 B | 0.013 | 0.02 | 5 439 488 B | 4 116 480 B | Da |
| 1 000 000 B | 0.146 | 0.1890 | 22 331 392 B | 12 230 656 B | Da |
| 3 233 680 B  (Bacteriovorax) | 0.844 | 0.6830 | 65 126 400 B | 32 350 208 B | Da |
| 5 277 677 B  (E. coli) | 1.664 | 1.1990 | 105 111 552B | 50 745 344 B | Da |
| 4 641 653 B  (E. coli) | 1.49 | 1.0530 | 93 573 120 B | 44969984 B | Da |

# Upute za instalaciju

1. Kompajlirati kod naredbom g++ SA-IS.cpp -o main.exe
2. Pokrenuti program naredbom ./main.exe ./imeDatoteke.txt , gdje je imeDatoteke.txt datoteka u kojoj piše genom s zadnjim znakom '$'.

Program stvara datoteke sa.txt i lcp.txt u trenutnom direktoriju.

# Literatura

[1] Domazet-Lošo, M., 5. predavanje: Sufiksno polje, Kolegij Bioinformatika, FER, Sveučilište u Zagrebu, 19.11.2019.

[2] Fischer J., Inducing the LCP-Array, 12th International Symposium, WADS 2011, New York, 2011., pp: 374-385

[3] Originalna implementacija: <http://algo2.iti.kit.edu/english/1828.php>

Proc. 12th Algorithms and Data Structures Symposium (WADS'11), Lecture Notes in Computer Science 6844, 374-385, Springer-Verlag, 2011.

[4] Fischer J., Kurpicz F., Dismantling DivSufSort, Prague Stringology Conference, Prag, 2017., pp: 62-76