Najduži zajednički podniz

Projektni zadatak iz kolegija *Oblikovanje i analiza algoritama*

Ružica Jović

20. 11. 2023.

Motivacija

- genetika - povezanost dvaju ili više organizama se često utvrđuje usporedbom DNA
- baze DNA lanca: adenin (A), citozin (C), gvanin (G) i timin (T)
- npr. $S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA$, $S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA$
- tražimo niz S_3 čiji su elementi baze koje se nalaze i u S_1 i u S_2, u istom poretku, ali ne nužno zaredom - S_1 i S_2 su sličniji što je S_3 duži

Definicija problema

- nizovi $X = \langle x_1, x_2, \ldots, x_m \rangle$ i $Z = \langle z_1, z_2, \ldots, z_k \rangle$ - Z je **podniz** (*eng. sequence*) od X ako postoji strogo rastući niz $\langle i_1, i_2, \ldots, i_k \rangle$ indeksa od X takav da za svaki $j \in \{1,2,\ldots,k\}$ vrijedi $z_j = x_{ij}$
  - npr. $Z = \langle B,C,D,B \rangle$ je podniz od $X = \langle A,B,C,B,D,A,B \rangle$ s pripadnim indeksima $\langle 2,3,5,7 \rangle$

- nizovi X, Y - niz Z je **zajednički podniz** (*eng. common sequence*) od X i Y ako je Z istovremeno podniz i od X i od Y
  - npr. $Z = \langle B,C,A \rangle$ je zajednički podniz od $X = \langle A,B,C,B,D,A,B \rangle$ i $Y = \langle B,D,C,A,B,A \rangle$

- za zadane nizove $X = \langle x_1, x_2, \ldots, x_m \rangle$ i $Y = \langle y_1, y_2, \ldots, y_n \rangle$ tražimo zajednički podniz $Z = \langle z_1, z_2, \ldots, z_k \rangle$ najveće duljine, odnosno **najduži zajednički podniz** (*eng. longest common subsequence,* skraćeno *LCS*)
- ne mora biti jedinstven
  - npr. $\langle B,C,B,A \rangle$ i $\langle B,D,A,B \rangle$ u prethodnom primjeru

- početni pristup - pronaći najdulji podniz od X koji je ujedno i podniz od Y
- X ima 2^m podnizova i za svaki je potrebno O(n) vremena da se provjeri je li podniz od Y - složenost **$O(2^m n)$**

- problem najdužeg zajedničkog podniza ima svojstvo "optimalne podstrukture"
- potproblemi odgovaraju prefiksima početnog niza, gdje **i-ti prefiks** niza $X = <x_1,x_2,\ldots,x_m>$ definiramo kao $X_i = <x_1,x_2,\ldots,x_i>$ za $i = 0,1,\ldots,m$
  - npr. za $X = <A,B,C,B,D,A,B>$, $X_4 = <A,B,C,B>$, $X_0 = \emptyset$

TEOREM Neka su $X = \langle x_1, x_2, \ldots, x_m \rangle$ i $Y = \langle y_1, y_2, \ldots, y_n \rangle$ zadani nizovi te $Z = \langle z_1, z_2, \ldots, z_k \rangle$ neki njihov najduži zajednički podniz.

(1) Ako je $x_m = y_n$, tada je $z_k = x_m = y_n$ i Z_{k-1} je najduži zajednički podniz od X_{m-1} i Y_{n-1}.

(2) Ako je $x_m \neq y_n$ i $z_k \neq x_m$, tada je Z najduži zajednički podniz od X_{m-1} i Y.

(3) Ako je $x_m \neq y_n$ i $z_k \neq y_n$, tada je Z najduži zajednički podniz od X i Y_{n-1}.

DOKAZ (1) Pretpostavimo da vrijedi $z_k \neq x_m$. Tada možemo dodati $x_m = y_n$ u Z, te dobivamo zajednički podniz od X i Y duljine k+1, što je u kontradikciji s pretpostavkom da je Z najduži zajednički podniz od X i Y. Nadalje, prefiks Z_{k-1} je zajednički podniz od X_{m-1} i Y_{n-1} duljine k–1. Pretpostavimo da postoji njihov zajednički podniz W duljine veće od k–1. Ako na njega dodamo $x_m = y_n$, dobivamo zajednički podniz od X i Y duljine veće od k, što je opet u kontradikciji s početnom pretpostavkom. Dakle, Z_{k-1} je najduži zajednički podniz od X_{m-1} i Y_{n-1}

(2) Pretpostavimo da postoji W zajednički podniz od X_{m-1} i Y duljine veće od k. Tad bi W ujedno bio i najduži zajednički podniz od X i Y, što je u kontradikciji s početnom pretpostavkom.

(3) Analogno kao (2).

c[i, j] = duljina najdužeg zajedničkog podniza za nizove X_i i Y_j

- po prethodnom teoremu vrijedi:

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j\,, \\ \max\{c[i, j-1], c[i-1, j]\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

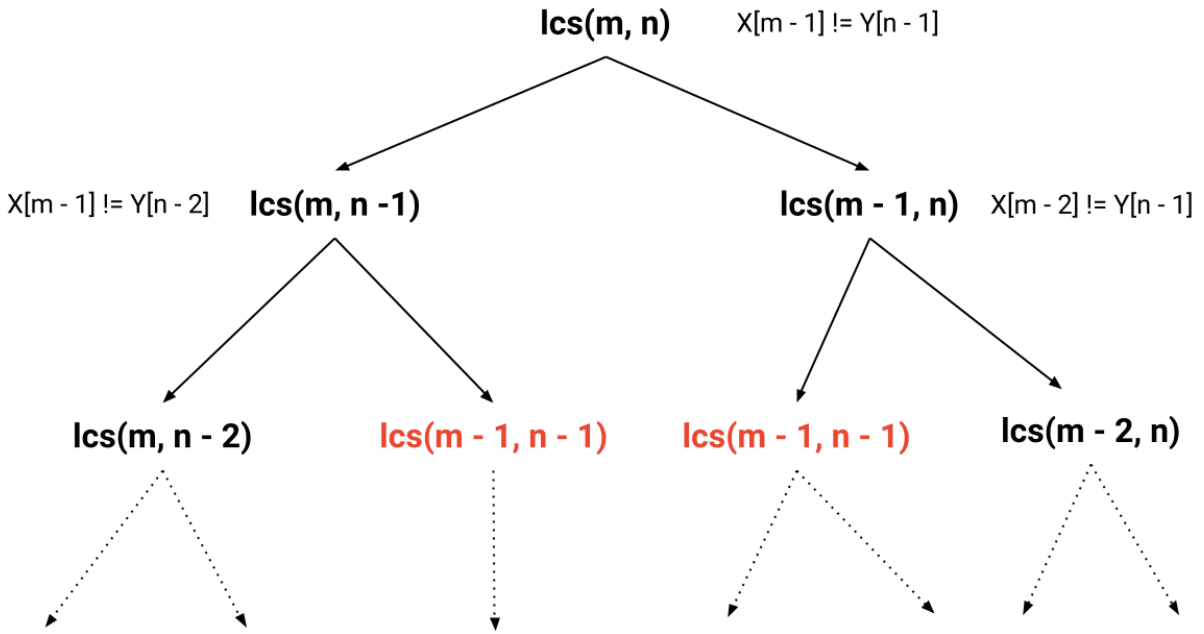
```
11  int max(int a, int b) {
12
13      return a >= b ? a : b;
14  }
15
16  int lcs_length_rec(char* X, char* Y, int i, int j) {
17
18      if(i == 0 || j == 0) {
19          return 0;
20      }
21
22      if(X[i - 1] == Y[j - 1]) {
23          return lcs_length_rec(X, Y, i - 1, j - 1) + 1;
24      }
25
26      return max(lcs_length_rec(X, Y, i - 1, j), lcs_length_rec(X, Y, i, j - 1));
27  }
```

- najbolji slučaj - elementi nizova X i Y se podudaraju sve dok ne iscrpimo jedan niz - $\min\{m, n\}$ poziva funkcije
- najgori slučaj - u svakom koraku računamo najduži zajednički podniz za potprobleme (i-1, j) i (i, j-1)
- X ima 2^m podnizova, a Y 2^n - složenost $O(2^m 2^m) = \mathbf{O(2^{m+n})}$

- *lcs_length_rec* računa samo duljinu najdužeg zajedničkog podniza pa implementiramo funkciju *lcs_rec* koja će dodatno pamtiti i sam najduži zajednički podniz
- ako x_i i y_j nisu jednaki - potrebna dodatna memorija veličine 2max{i, j} za spremanje rješenja potproblema

```
29  void lcs_rec(char* X, char* Y, int i, int j, char* result, int* result_length) {
30
31      if (i == 0 || j == 0) {
32          *result_length = 0;
33      } else if (X[i - 1] == Y[j - 1]) {
34          lcs_rec(X, Y, i - 1, j - 1, result, result_length);
35          (*result_length)++;
36          result[*result_length - 1] = X[i - 1];
37      } else {
38          char *result1, *result2;
39          int result1_length, result2_length, k;
40          k = max(i, j);
41          result1 = (char*) malloc(k * sizeof(char));
42          result2 = (char*) malloc(k * sizeof(char));
43
44          lcs_rec(X, Y, i - 1, j, result1, &result1_length);
45          lcs_rec(X, Y, i, j - 1, result2, &result2_length);
46
47          if(result1_length >= result2_length) {
48              memcpy(result, result1, result1_length);
49              *result_length = result1_length;
50          } else {
51              memcpy(result, result2, result2_length);
52              *result_length = result2_length;
53          }
54
55          free(result1);
56          free(result2);
57      }
58  }
```

- postoje potproblemi koji se preklapaju
- problem ima samo mn različitih potproblema - računanje c[i, j] za 1<=i<=m i 1<=j<=n
- dinamičko programiranje - vrijednosti c[i, j] spremamo u memoriju
- *top-down* i *bottom-up* pristup



Top-down pristup

- vrijednosti c[i, j] inicijaliziramo na –1
- vrijednosti c[i, j] računamo rekurzivno, počevši od početnog problema c[m, n]

- složenost: $\Theta(mn)$
- potrebna dodatna memorija veličine mn za spremanje vrijednosti c[i, j]

```
60  int lcs_length_rec_top_down(char* X, char* Y, int i, int j, int** C) {
61
62      if(i == 0 || j == 0) {
63          return 0;
64      }
65
66      if(C[i - 1][j - 1] != -1) {
67          return C[i - 1][j - 1];
68      }
69
70      if(X[i - 1] == Y[j - 1]) {
71          return C[i - 1][j - 1] = lcs_length_rec_top_down(X, Y, i - 1, j - 1, C) + 1;
72      }
73
74      return C[i - 1][j - 1] = max(lcs_length_rec_top_down(X, Y, i - 1, j, C), lcs_length_rec_top_down(X, Y, i, j - 1, C));
75  }
76
77  int lcs_length_top_down(char* X, char* Y, int m, int n) {
78
79      int **C, i, j;
80
81      C = (int**) malloc(m * sizeof(int*));
82      for(i = 0; i < m; i++) {
83          C[i] = (int*) malloc(n * sizeof(int));
84      }
85
86      for(int i = 0; i < m; i++) {
87          for(j = 0; j < n; j++) {
88              C[i][j] = -1;
89          }
90      }
91
92      int result = lcs_length_rec_top_down(X, Y, m, n, C);
93
94      for(i = 0; i < m; i++) {
95          free(C[i]);
96      }
97      free(C);
98
99      return result;
100 }
```

Bottom-up pristup

- potrebna je matrica dimenzija (m+1) x (n+1)
- prvi red i stupac inicijaliziramo na 0
- krećemo od c[1, 1] i popunjavamo vrijednosti c[i, j] red po red
- rješenje početnog problema je na poziciji c[m, n]

- složenost: $\Theta(mn)$
  - efikasniji nego *top-down* pristup jer samo jednom prolazimo kroz matricu C

```
102 int lcs_length_bottom_up(char* X, char* Y, int m, int n) {
103
104     int **C, i, j;
105
106     C = (int**) malloc((m + 1) * sizeof(int*));
107     for(i = 0; i < m + 1; i++) {
108         C[i] = (int*) malloc((n + 1) * sizeof(int));
109     }
110
111     for(i = 0; i <= n; i++) {
112         C[0][i] = 0;
113     }
114     for(i = 0; i <= m; i++) {
115         C[i][0] = 0;
116     }
117
118     for(i = 1; i <= m; i++) {
119         for(j = 1; j <= n; j++) {
120             if(X[i - 1] == Y[j - 1]) {
121                 C[i][j] = C[i - 1][j - 1] + 1;
122             } else if(C[i - 1][j] >= C[i][j - 1]) {
123                 C[i][j] = C[i - 1][j];
124             } else {
125                 C[i][j] = C[i][j - 1];
126             }
127         }
128     }
129
130     int result = C[m][n];
131
132     for(int i = 0; i < m + 1; i++) {
133         free(C[i]);
134     }
135     free(C);
136
137     return result;
138 }
```

Konstrukcija najdužeg zajedničkog podniza

- pomoću dodatne matrice dimenzija m x n koja će sadržavati vrijednosti b[i, j]
  - b[i, j] predstavljaju smjer ("dijagonalno iznad", "iznad" ili "lijevo") koji pokazuje na optimalan potproblem - računaju se kod računanja vrijednosti c[i, j]
    1. $x_i = y_j$ -> "dijagonalno iznad"
    2. c[i-1, j] >= c[i, j-1] -> "iznad"
    3. c[i, j-1] >= c[i-1, j] -> "lijevo"
- najduži zajednički podniz (u obrnutom poretku) konstruiramo počevši od vrijednosti b[m, n] te slijedimo smjer zapisan u b[i, j] dok ne dođemo do i = 0 ili j = 0
- potrebno mn dodatne memorije i **dodatno O(m+n)** vremena (u svakom pozivu funkcija dekrementira barem jedan od i i j)

```
140 void lcs_construction(char* X, int** B, int i, int j, char* result, int result_position) {
141
142     if(i >= 0 && j >= 0) {
143         switch(B[i][j]) {
144             case DIAGONAL:
145                 result[result_position--] = X[i];
146                 lcs_construction(X, B, i - 1, j - 1, result, result_position);
147                 break;
148             case UP:
149                 lcs_construction(X, B, i - 1, j, result, result_position);
150                 break;
151             case LEFT:
152                 lcs_construction(X, B, i, j - 1, result, result_position);
153         }
154     }
155 }
```

X = <ABCBDAB>, Y = <BDCABA>

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

- u posljednjem *bottom-up* algoritmu za pronalazak najdužeg zajedničkog podniza, niz Z možemo rekonstruirati i bez korištenja dodatne memorije, pomoću vrijednosti c[i, j]
- za zadani c[i, j] možemo u O(1) vremenu odrediti koja od vrijednosti c[i-1, j-1], c[i – 1, j] i c[i, j-1] je korištena za računanje c[i, j]
- ušteda Θ(mn) memorije - ne poboljšava značajno efikasnost algoritma jer nam je već za matricu C potrebno Θ(mn) memorije

- za računanje c[i, j] potrebna samo dva reda - možemo u potpunosti eliminirati matricu C
  - korisno samo za računanje duljine najdužeg zajedničkog podniza (bez matrice C nemamo dovoljno informacija za konstrukciju niza u vremenu O(m+n))

Empirijska analiza algoritama

- programski jezik C, operacijski sustav Linux Ubuntu 22.04, procesor i5-8265U, 1.60 GHz, 8 GB RAM-a
- elementi nizova su znakovi iz skupa {A, B, C, D, E, F, G, H, I, J}, nizovi su jednake duljine
- testove izvršavamo 5 puta za različite nasumično generirane nizove znakova

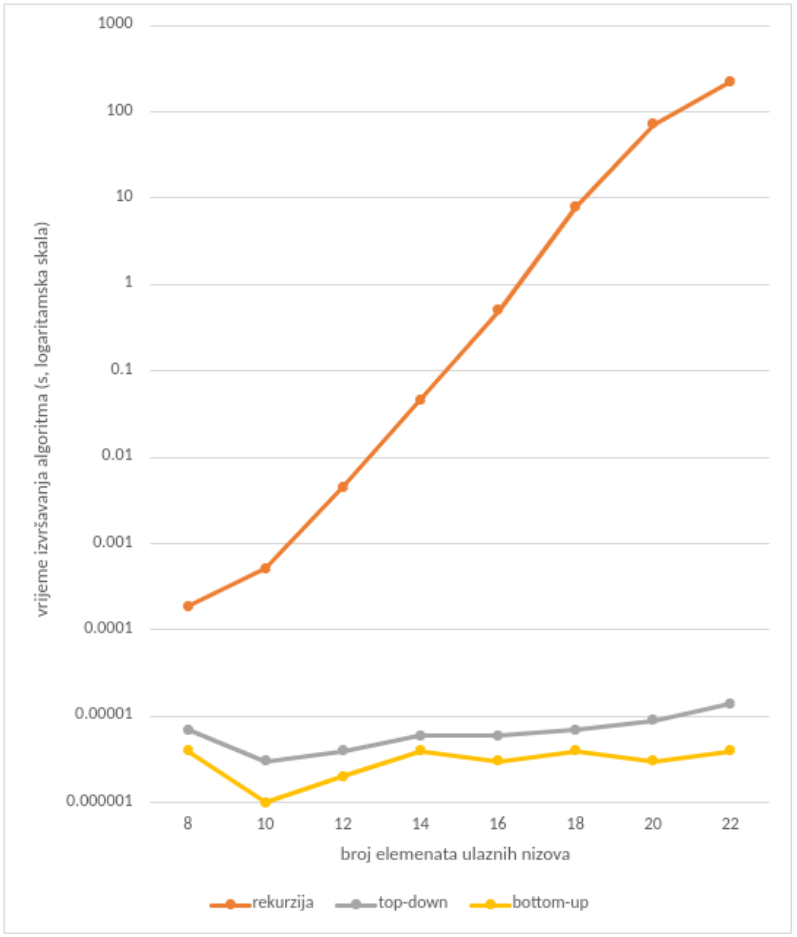
Algoritmi za računanje duljine najdužeg zajedničkog podniza

broj elemenata ulaznih nizova	vrijeme izvršavanja algoritma (s)		
	rekurzija	top-down	bottom-up
8	0,000186	0,000007	0,000004
10	0,000517	0,000003	0,000001
12	0,004536	0,000004	0,000002
14	0,046181	0,000006	0,000004
16	0,497992	0,000006	0,000003
18	8,0638	0,000007	0,000004
20	71,223	0,000009	0,000003
22	224,84139	0,000014	0,000004

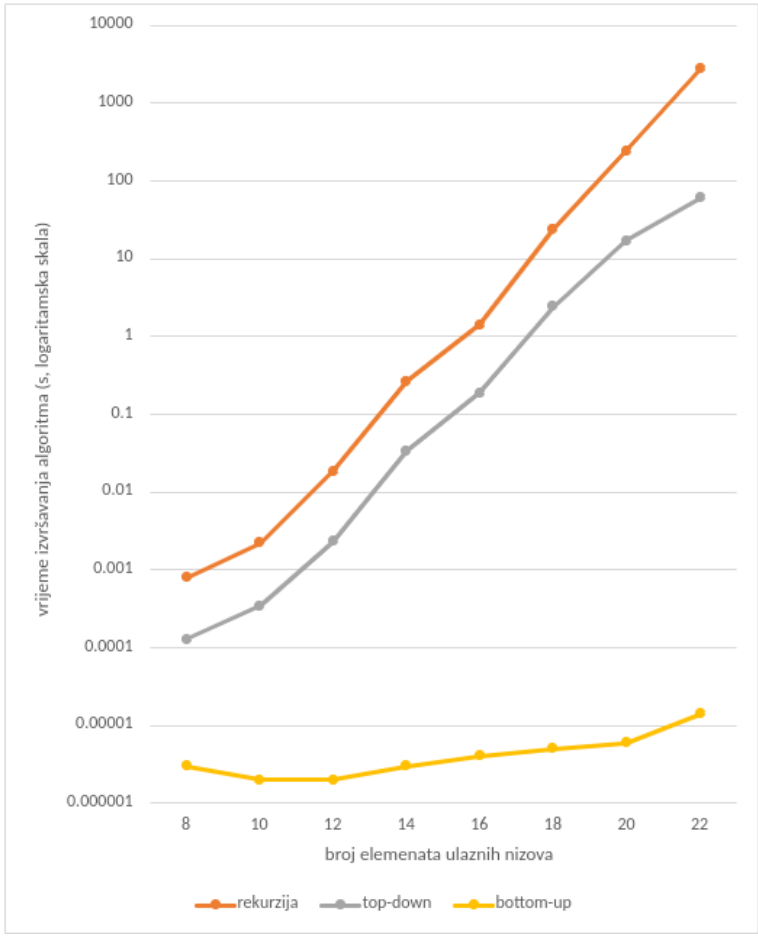
Algoritmi za pronalazak najdužeg zajedničkog podniza

broj elemenata ulaznih nizova	vrijeme izvršavanja algoritma (s)		
	rekurzija	top-down	bottom-up
8	0,000793	0,000127	0,000003
10	0,00224	0,000343	0,000002
12	0,018519	0,002293	0,000002
14	0,267247	0,033648	0,000003
16	1,425799	0,187883	0,000004
18	24,03086	2,435424	0,000005
20	243,0111	16,98837	0,000006
22	2758,340023	60,629094	0,000014

Algoritmi za računanje duljine najdužeg zajedničkog podniza

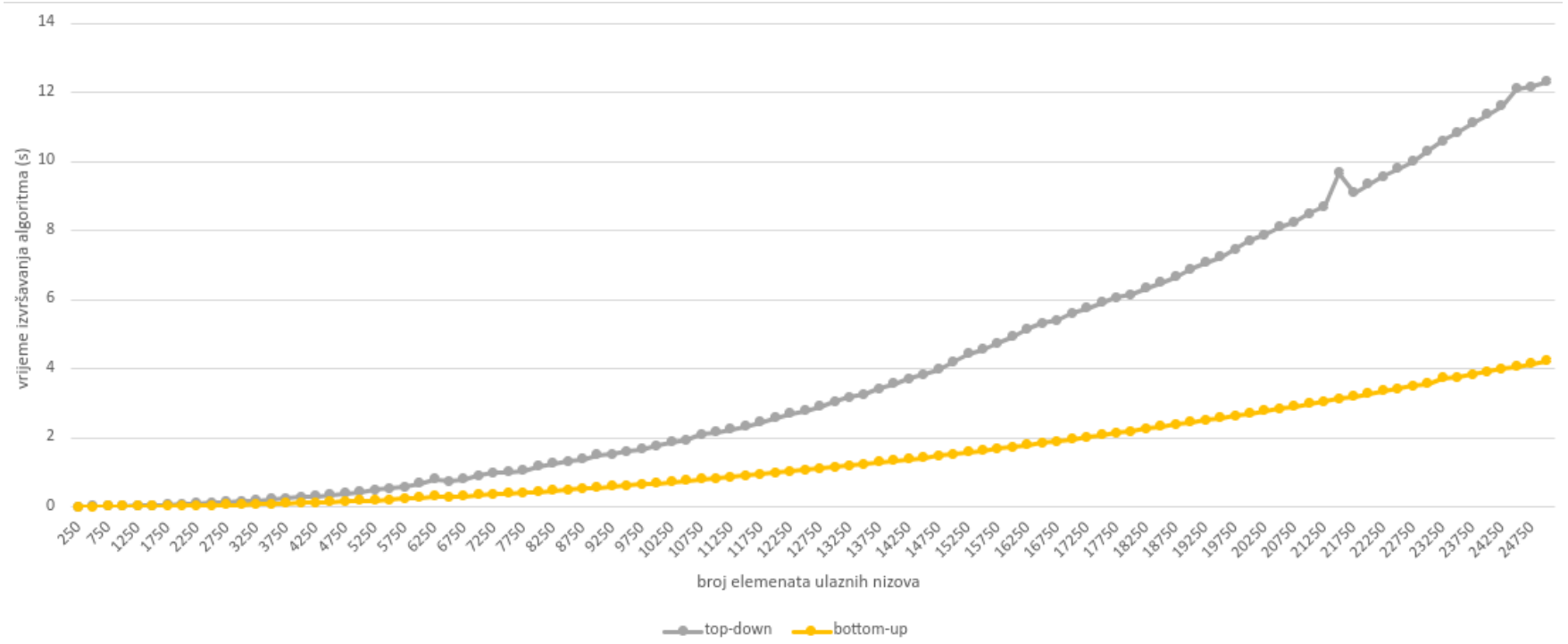


Algoritmi za pronalazak najdužeg zajedničkog podni za



*logaritamske skale

Usporedba algoritama koji koriste dinamičko programiranje



broj elemenata ulaznih nizova	vrijeme izvršavanja algoritma (s)	
	top-down	bottom-up
250	0.001885	0.00056
5000	0.431594	0.17204
10000	1.761738	0.680057
15000	4.194859	1.524104
20000	7.700611	2.704069
25000	12.297199	4.219907

Hvala na pažnji!

Literatura

- T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to algorithms 4th edition*, The MIT Press, 2022.
- M. T. Goodrich, R. Tamassia, *Algorithm design and applications*, Wiley, 2015.