UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Miloš P. Miković

ALGORITMI ZA REŠAVANJE PROBLEMA NAJKRAĆE ZAJEDNIČKE NADNISKE

master rad

Beograd, 2021.

Mentor:

dr Aleksandar KARTELJ, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Vladimir FILIPOVIĆ, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Stefan MIŠKOVIĆ, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Hvala profesoru Aleksandru Kartelju.

Naslov master rada: Algoritmi za rešavanje problema najkraće zajedničke nadniske

Rezime:

Ključne reči: optimizacija, pretraga bima, analiza bioloških sekvenci

Sadržaj

Glava 1

Uvod

Problem najkraće zajedničke nadniske (*eng.* Shortest Common Supersequence Problem) jedan je od dobro poznatih NP-teških problema optimizacije u oblasti analize reči [1]. Ukratko, PNZN[1] se može opisati kao problem pronalaženja najkraće reči ω sačinjene od simbola zadate konačne Azbuke Σ, tako da su sve sekvence iz unapred zadatog konačnog skupa $\mathcal{L}$ sadržane u sekvenci ω. Kada se kaže da su sve reči iz skupa $\mathcal{L}$ sadržane, misli se na to da se svaka reč iz skupa $\mathcal{L}$ može dobiti uklanjanjem simbola iz reči ω ali u zadatom redosledu [3]. PNZN ima primene u mnogim oblastima informatike uključujući kompresiju podataka [4], optimizaciju upita [13], analizu i poređenje teksta i bioloških sekvenci, [15] [2] kao i bioinformatiku [1]. Kao rezultat velike primene u mnogim oblastima, postoji veliki broj istraživanja na temu ovog problema u pokušaju da se dođe do što boljeg i prihvatljivijeg rešenja. Trenutno najbolji algoritmi za rešavanje PNZN počivaju na metaheurističkoj metodi pretraga bima (*eng.* beam search) koja će biti predstavljena u ovom radu. U nastavku uvodnog poglavlja biće formalno definisan problem najkraće zajedničke nadniske i biće dat pregled dosadašnjih istraživanja na temu ovog problema.

1.1 Problem najkraće zajedničke nadniske

U ovom poglavlju formalno ćemo definisati PNZN, ali pre toga uvešćemo potrebnu notaciju koja će biti korišćena u nastavku teksta. Konačna azbuku sastoji se od konačnog broja slova i označavaćemo je sa Σ. Svaka konačna reč $\omega = \omega(1)\omega(2)...\omega(n)$ sastoji se od konačnog broja slova azbuke gde $\omega(j) \in \Sigma$ predstavlja j-to slovo reči

[1]U nastavku teksta PNZN ćemo koristiti kao skraćenicu za problem najkraće zajedničke nadniske

$\omega \in \Sigma^*$. Duzinu reči ω označavaćemo sa $|\omega|$, praznu reč sa ε i važi da $|\varepsilon| = 0$. Reč koja se dobija dodavanjem slova α na početak reči ω označavaćemo sa $\alpha\omega$, a reč koja se dobija dodavanjem slova α na kraj reči ω sa $\omega\alpha$ Sa $\omega[a : b]$[2] označavaćemo reč koja se dobija od reči ω počevši od pozicije a do pozicije b u reči ω. Ako se izostavi vrednost pozicije a, podrazumeva se da je početna pozicija 0, slično ako se izostavi vrednost pozicije b podrazumeva se vrednost $m - 1$ ako $|\omega| = m$. Na primer $\omega[:]$ predstavlja samu reč ω. Pristupanje slovu na poziciji i reči ω označavaćemo sa $\omega[i]$.

Neka važi da $\omega_1, \omega_2 \in \Sigma^*$, za reč ω_1 kažemo da je supersekvenca reči ω_2 u oznaci $\omega_1 \succ \omega_2$ ako važi sledeća rekurzivna definicija [1]:

$$\begin{aligned} \omega_1 \succ \varepsilon &\triangleq \text{Tačno} \\ \varepsilon \succ \omega_2 &\triangleq \text{Netačno}, \quad \text{Ako } \omega_2 \neq \varepsilon \\ \alpha\omega_1 \succ \alpha\omega_2 &\triangleq \omega_1 \succ \omega_2 \\ \alpha\omega_1 \succ \beta\omega_2 &\triangleq \omega_1 \succ \beta\omega_2, \quad \text{Ako } \alpha \neq \beta \end{aligned} \tag{1.1}$$

Zapravo, $\omega_1 \succ \omega_2$ označava da se svi simboli iz ω_2 nalaze u ω_1 u datom redosledu, ali ne nužno uzastopno. Na primer, za datu azbuku $\Sigma = \{a, c, t, g\}$, važi $agcatg \succ act$. Sada možemo formalno definisati PNZN. Instanca PNZN može se definisati kao $\mathcal{I} = (\Sigma, \mathcal{L})$, gde Σ predstavlja konačnu azbuku, a $\mathcal{L}$ predstavlja skup od m reči $\{\omega_1, \omega_2, ..., \omega_m\}$, $\omega_i \in \Sigma^*$. Potrebno je pronaći reč ω najmanje dužine tako da važi da je ω supersekvenca svake reči iz skupa $\mathcal{L}$ ($\omega \succ \omega_i, \forall\omega_i \in \mathcal{L}$ i $|\omega|$ je minimalna). Na primer za instancu PNZN $\mathcal{I} = (\{a, c, t, g\}, \{act, cta, aca\})$, najmanja zajednička supersekvenca instance $\mathcal{I}$ je $acta$.

Može se pokazati da je PNZN NP-težak problem, čak i ako su jaka ograničenja postavljena na $\mathcal{L}$ ili Σ. Dokazano je da je PNZN NP-kompletan problem nad svakom azbukom Σ za koju važi da $|\Sigma| \geqslant 2$ [8] ili kada su sve reči $\omega \in \mathcal{L}$ dužine dva [16]. U principu ovim rezultatima NP-težine se mora pristupiti sa oprezom, jer predstavljaju samo najgori scenario što često u praksi nije slučaj. U odnosu na to, realnija karakterizacija težine može se dobiti korišćenjem okvira parametrizovane složenosti (*eng.* framework of parameterized complexity). Ukratko, ovo se postiže višedimenzionim pristupom problemu, shvatanjem njegove unutrašnje strukture i izolovanjem određenih parametara. Ako se težina (koja nije polinomijalna) može izolovati unutar

[2]Slično kao indeksiranje nizova u programskom jeziku Python

ovih parametara, problem može biti efikasno rešen za fiksne vrednosti ovih parametara. Na primer može se uzeti maksimalna dužina k nadniske kao parametar i ako je pritom veličina azbuke fiksna ili parametar takođe, problem postaje fiksno-parametarski pratljiv (*eng.* fixed-parameter tractable) jer postoji maksimalno $|\Sigma|^k$ nadniski koje se mogu proveriti kao rešenje problema [1]. Može se parametrizovati i broj reči u skupu $\mathcal{L}$.

1.2 Pregled dosadašnjih istraživanja

Problem najkraće zajedničke nadniske prvi je uveo Dejvid Mejer (*eng.* David Maier) 1978. godine u svom radu „The Complexity of Some Problems on Subsequences and Supersequences" [9]. Korišćenjem dinamičkog programiranja (*eng.* dynamic programming) PNZN nad dve reči dužine n rešen je algoritmom vremenske složenosti $\mathcal{O}(n^2)$ i prostorne složenosti $\mathcal{O}(n^2)$. Algortiam zasnovan na dinamičkom programiranju može biti unapređen, pa tako za k reči dužine maksimalno n, PNZN može biti rešen u $\mathcal{O}(n^k)$ prostornoj i vremenskoj složenosti [14]. Jasno je da ovakav algoritam nije praktičan za velike vrednosti k. S obzirom na to da ne postoji algoritam polinomijalne složenosti koji rešava PNZN, pribegava se optimizacionim metodama u rešavanju ovog problema. Ono što je karakteristično za optimizacioni pristup rešavanju problema jeste to da se formira algoritam koji rešava postojeći problem tako što daje rešenje koje je prihvatljivo pod određenim uslovima. Takvo rešenje ne mora nužno biti optimalno rešenje problem. Na ovaj način, korišćenjem određene optimizacione tehnike, dobija se algoritam koji se izvršava brzo u realnim uslovima i daje prihvatljivo dobra rešenja.

Vremenom je predloženo mnogo heurističkih i metaheurističkih algoritama za rešavanje PNZN. Neke od poznatijih heurističkih funkcija koje su korišćene u rešavanju PNZN su Alfabet (*eng.* Alphabet) [10], Većinsko Spajanje (*eng.* Majority Merge) i Težinsko Većinsko Spajanje (*eng.* Weighted Majority Merge) [1], Turnirska (*eng.* Tournament) i Pohlepna (*eng.* Greedy) [17], Redukuj-Proširi (*eng.* Reduce-Expand) [10]. Pored navedenih funkcija, korišćeni su i metaheuristički algoritmi, genetski algoritam (*eng.* genetic algorithm) [6] i optimizacija kolonijom (*eng.* colony optimization) [12], koji predstavljaju složenije optimizacione tehnike i imaju tendenciju ka dužem vremenu izvršavanja ne većim instancama problema [7].

Jedna od trenutno najboljih metaheuristika za rešavanje PNZN jeste pretraga bima (*eng.* beam search). Ukratko, pretraga bima predstavlja nepotpunu pretragu

stabla, koja na svakom nivou proširuje graf stanja tako što napreduje sa čvorovima koji najviše obećavaju [5]. Upravo zbog toga što se dobro pokazala u rešavanju PNZN i sličnih problema poput problema najduže zajedničke podniske (*eng.* longest common subsequence) pretraga bima će biti korišćena u ovom radu i biće detaljnije opisana u daljem tekstu. S obzirom na to da pretraga bima podrazumeva postojanje heurističke funkcije koja će oceniti kvalitet čvorova na određenom nivou, u ovom radu izabrane su dve takve funkcije, Većinsko Spajanje (*eng.* Majority Merge) i Težinsko Većinsko Spajanje (*eng.* Weighted Majority Merge) koje su se dobro pokazale u prethodnim istraživanjima i one će detaljnije biti opisane u nastavku teksta.

Glava 2

Algoritmi za rešavanje problem najkraće zajedničke nadniske

U ovom poglavlju biće dat pregled sledeća dva algoritama koja su korišćena za rešavanje problema najkraće zajedničke nadniske:

1. Algoritam grananja sa odsecanjem (*eng.* backtracking)
2. Pretraga Bima (*eng.* Beam Search)

U nastavku teksta biće opisana konstrukcija ova dva algoritma kao i njihove karakteristike.

2.1 Algoritam grananja sa odsecanjem

Algoritam grananja sa odsecanjem poboljšava tehniku grube sile tako što vrši provere tokom generisanja kandidata za rešenja i tako što se odbacuju parcijalno popunjeni kandidati za koje se unapred može utvrditi da se ne mogu proširiti do optimalnog rešenja problema. Dakle, grananje sa odsecanjem podrazumeva da se tokom obilaska u dubinu drveta, kojim se predstavlja prostor potencijalnih rešenja odsecaju oni delovi drveta za koje se unapred može utvrditi da ne sadrže ni jedno rešenje problema tj. da ne sadrže optimalno rešenje, pri čemu se odsecanje vrši i u čvorovima bliskim korenu koji mogu da sadrže i samo parcijalno popunjene kandidate za rešenja. Dakle, umesto da se čeka da se tokom pretrage stigne do lista (ili eventualno unutrašnjeg čvora koji predstavlja nekog kandidata za rešenje) i da se provera zadovoljenosti uslova ili optimalnosti vrši tek tada, prilikom granja

sa odsecanjem provera se vrši u svakom koraku i vrši se provera parcijalno popunjenih rešenja. Efikasnost ovakvog algoritma uveliko zavisi od kvaliteta kriterijuma na osnovu kojih se vrši odsecanje. Iako obično složenost najgoreg slučaja ostaje eksponencijalna (kakva je po pravili kod algoritama grube sile), pažljivo odabrani kriterijumi odsecanja mogu odseći jako velike delove pretrage (koji su često takođe eksponencijalne veličine u odnosu na dimenzije ulaznog problema) i time značajno ubrzati proces pretrage.

U kontekstu PNZN korišćen je rekurzivni algoritam prikazan pod Algoritam 1. Algoritam podrazumeva postojanje dve globalne promenjive $minD$ i $maxD$ čije se

Algoritam 1 GrananjeSaOdsecanjem(ω)

```
1: d ← |ω|                                          ▷ d - dužina trenutne reči
2: if (d > maxD) ∨ (d ≥ nd) then
3:     return
4: end if
5: if (d ⩾ minD) ∧ ZajedničkaNadniska(ω) then
6:     iDaljeNadniska ← Tačno
7:     pozicija ← 0
8:     while iDaljeNadniska do                              ▷ optimizacija
9:         if ZajedničkaNadniska(ω[pozicija + 1 :]) then
10:            pozicija ← pozicija + 1
11:        else
12:            iDaljeNadniska = Netačno
13:        end if
14:    end while
15:    nn ← ω[pozicija :]                          ▷ nn - najbolja nadniska
16:    nd ← |nn|                                   ▷ nd - najbolja dužina
17: end if
18: for α ∈ Σ do                                   ▷ grananje po slovima azbuke
19:    GrananjeSaOdsecanjem(ωα)
20: end for
```

vrednosti postavljaju pre poziva algoritma i one označavaju redom minimalnu i maksimalnu dubinu, takođe se podrazumeva da su skupovi $\mathcal{L}$ i Σ globalno dostupni u programu. Vrednost promenjive $minD$ postavlja se na dužinu najkraće reči u $\mathcal{L}$, a vrednost promenjive $maxD$ predstavlja proizvod veličine azbuke i najduže reči u skupu $\mathcal{L}$ kao što je prikazano u jednačinama 2.1 i 2.2

$$minD = \{min_{\omega \in \mathcal{L}} |\omega|\} \tag{2.1}$$

$$maxD = |\Sigma| * L, \text{ gde } L = \{max_{\omega \in \mathcal{L}} |\omega|\} \tag{2.2}$$

Sada ćemo ova uvedena ograničenja minimalne i maksimalne dubine objasniti na kratkom primeru. Za instancu PNZN $\mathcal{I} = (\{a, c, t, g\}, \{acta, cta, ac\})$ znamo da najkraća zajednička nadniska ne može biti kraća od dužine najkraće reči u skupu $\mathcal{L}$, jer takva niska ne bi sadržala ni minimalnu reč a samim time ni ostale reči u $\mathcal{L}$, pa vrednost promenjive $minD = |ac| = 2$. Takođe za instancu problema I mi sigurno znamo jedno rešenje PNZN. Kako je najduža reč $acta$ dužine 4 važi da reč $\omega = \underline{actg_1}\,\underline{actg_2}\,\underline{actg_3}\,\underline{actg_4}$ sigurno predstavlja jedno rešenje problema, pa važi da $maxD = |\omega| = |\Sigma| * |acta| = 4 * 4 = 16$.

Nakon postavljanja ograničenja dubine, rekurzivni algoritam se poziva u obliku GrananjeSaOdsecanjem(ε) kako bi pretraga u dubinu krenula od prazne reči. U svakom koraku algoritma vrši se provera da li je dužina trenutne reči d veća od maksimalne dubine ili je veća od dužine trenutno najbolje pronađene nadniske (na početku $nd = maxD + 1$) i ako jeste vrši se odsecanje tog podstabla u prostoru pretrage. Ako važi da je dužina trenutne reči d veća od minimalne dubine i pritom važi da reč ω predstavlja zajednučku nadnisku reči u skupu $\mathcal{L}$, pronađeno je jedno rešenje PNZN. Pre ažuriranje vrednosti najbolje nadniske (nn) i najbolje dužine (nd) pokušava se sa skraćivanjem reči ω u cilju dobijanja kraće nadniske, a time i većeg broja odsecanja u prostoru pretrage. Na primeru gore navedene instance problema I, kako se obilazak stabla vrši u dubinu i grananje u svakom rekurzivnom pozivu počinje prvim slovom azbuke, u ovom slučaju a, često se dobijaju rešenja oblika $\omega = aa...aR$, gde ω predstavlja rešenje PNZN, ali i R predstavlja rešenje pa stoga možemo ukloniti prefiks reči ω, sve dok skraćivanjem i dalje dobijamo validno rešenje problema. Na kraju, vrši se grananje po svim slovima azbuke i rekurzivno poziva algoritam sa argumentom $\omega\alpha$ koji proširuje trenutnu reč ω novim slovom iz azbuke Σ. Funkcija ZajedničkaNadniska(ω) vrši proveru da li reč ω predstavlja zajednučku nadnisku reči iz skupa $\mathcal{L}$, i u ovom slučaju se podrazumeva da je skup $\mathcal{L}$ globalno dostupan. Pseudo kod je dat pod Algoritam 2. U svakom koraku petlje proverava se da li je trenutna reč $s \in \mathcal{L}$ podniska reči ω, ako nije funkcija vraća Netačno, a ako su sve reči podniske reči ω funkcija vraća Tačno. U suštini važi jednostavna logička formula 2.3:

$$\{Podniska(s, \omega) \mid \forall s \in \mathcal{L}\} \Longleftrightarrow \{\omega \succ s \mid \forall s \in \mathcal{L}\} \tag{2.3}$$

Ako su sve reči iz skupa $\mathcal{L}$ podniske reči ω onda važi da je ω nadniska svih reči iz $\mathcal{L}$ i obrnutno. Pseudo kod funkcije Podniska(s, ω) dat je pod Algoritam 3. Na početku se vrši inicijalizacija promenjivih, sP i ωP predstavljaju indekse početka reči s i ω, sK i ωK redom predstavljaju indekse na kraj reči s i ω. U svakom koraku petlje vrši se provera da li se stiglo do kraja reči ω, i ako je to slučaj, a prethodno nismo prošli sva slova reči s znamo da s sigurno nije podniska reči ω. Svaki put kada se vrednosti na pozicijama sP i ωP u rečima s i ω poklope, oba indeksa inkrementiramo, a u suprotnom napredujemo samo kroz reč ω pa inkrementiramo njen indeks ωP. Petlja se završava kada prođemo sva slova reči s i tada znamo da je s sigurno podniska reči ω.

Algoritam 2 ZajedničkaNadniska(ω)

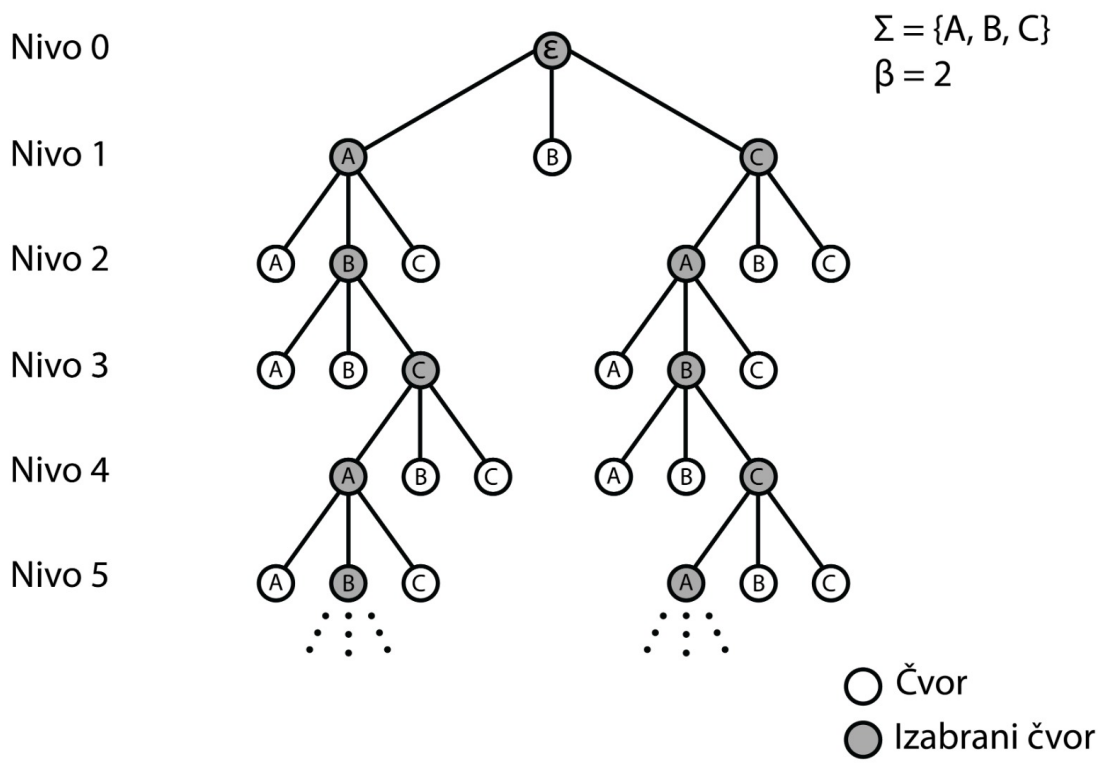
```
1: for s ∈ L do
2:     if ¬Podniska(s, ω) then
3:         return Netačno
4:     end if
5: end for
6:
7: return Tačno
```

Algoritam 3 Podniska(s, ω)

```
1: sP ← 0                          ▷ indeks na početak reči s
2: sK ← |s|                        ▷ indeks na kraj reči s
3: ωP ← 0                          ▷ indeks na početak reči ω
4: ωK ← |s|                        ▷ indeks na kraj reči ω
5:
6: while sP ≠ sK do
7:     if ωP == ωK then
8:         return Netačno
9:     else if s[sP] == ω[ωP] then
10:        sP ← sP + 1
11:        ωP ← ωP + 1
12:    else
13:        ωP ← ωP + 1
14:    end if
15: end while
16:
17: return Tačno
```

Iako algoritam grananja sa odsecanjem u najgorem slučaju ima eksponencijalnu složenost $O(|\Sigma|^{maxD})$ kao i algoritam Iscrpne Pretrage (*eng.* Brute Force), u praksi je prosečno vreme izvršavanja daleko kraće. Rezultati izvršavanja algoritma grananja sa odsecanjem biće prikazani u sledećem poglavlju. Ono što je bitno istaći jeste da ovaj algoritam garantuje pronalaženje optimalnog rešenja PNZN. S obzirom na eksponencijalnu prirodu ovog algoritma, ne moguće je koristiti ga za proveru kvaliteta rešenja Pretrage Bima na većim instancama problema, stoga će u te svrhe biti korišćene nešto sofisticiranije metode koje će biti objašnjene u sekciji 3.1.

2.2 Pretraga Bima



Slika 2.1: Šema metaheuristike Pretraga Bima

Pretraga Bima predstavlja metaheuristiku koja je uvedena 1976. godine u oblasti prepoznavanja govora (*eng.* speech recognition). Korišćena je u završnim slojevima mnogih modela za obradu prirodnog jezika (*eng.* natural language processing models) u donošenju odluke da se izabere najbolji izlaz date ciljne promenjive [11].

Pored navedenih primena pretraga bima se intenzivno koristi u sledećeim oblastima: kombinatorna optimizacija (*eng.* combinatorial optimization), problemi planiranja (*eng.* scheduling problems), problemi rutiranja vozila (*eng.* vehicle routing problems), podešavanje hiperparametara u oblasti mašinskog učenja (*eng.* Machine Learning Hyperparameter Tuning), problemi zadovoljenja ograničenja (*eng.* Constraint Satisfaction Problems).

Pretraga Bima predstavlja vrstu pretrage grafa u širinu u cilju da se pronađe najbolji put od korenog čvora do ciljanog čvora u grafu. Kako bi se složenost izračunavanja držala u zadatim granicama, Pretraga Bima evaluira dostignute čvorove na određenom nivou ali bira samo podskup od β čvorova koji najviše obećavaju i sa tim podskupom čvorova napreduje dalje u pretrazi. Izabrani podskup od β čvorova zvaćemo bim (*eng.* beam) i označavaćemo ga sa $\mathcal{B}$, a β parametar ćemo zvati širina bima (*eng.* beam width) [5]. U kontekstu PNZN graf koji pretražujemo $\mathcal{G} = (\mathcal{V}, \mathbb{E})$ predstavlja usmeren aciklički graf, a pseudo kod algoritma je prikazan pod Algoritam 4.

Kao i u prethodno opisanom algoritmu podrazumeva se da je promenjiva $maxD$ globalno dostupna, kao i skupovi $\mathcal{L}$ i Σ. Svaki element skupa $\mathcal{B}$ predstavlja objekat koji sadrži parcijalnu reč, heurističku ocenu i pozicije. Parcijalna reč predstavlja put od korena do nekog čvora unutar stabla pretrage, heuristička ocena predstavlja ocenu ove reči od strane heurističke funkcije, a pozicije predstavljaju niz koji ozančava pozicije do kojih se stiglo unutar skupa $\mathcal{L}$. Na primer ako je data azbuka $\Sigma = \{a, c, t, g\}$ i skup $\mathcal{L} = \{act, gta, gcc\}$ tada pozicije $p = [0, 0, 0]$ predstavljaju baš skup $\mathcal{L}$ odnosno pozicije na početak svake reči u $\mathcal{L}$, a pozicije $p^{'} = [1, 1, 2]$ predstavljaju skup $\mathcal{L}^{'} = \{ct, gt, c\}$ koji je dobijen od skupa $\mathcal{L}$ pomeranjem za vrednosti pozicija $p^{'}$. Pri proširivanju skupa $\mathcal{B}$ slovima iz azbuke Σ, dobijaju se novi elementi čija je parcijalna reč proširena novododatim slovom. Od jednog elementa dobijamo maksimalno $|\Sigma|$ novih, što znači da u svakom koraku algoritma dobijamo maksimalno $\beta * |\Sigma|$ novih elemenata od kojih je potrebno izabrati β elemenata. Heurističke funkcije, koje će biti predstavljene u narednom podpoglavlju, za dati niz pozicija računaju težine za svako slovo azbuke, koje se zatim pri proširivanju bima dodaju svakom elementu koji se proširi odgovarajućim slovom. Ovim se eliminiše potreba za ocenjivanjem svakog elementa bima iznova i iznova na svakom nivou pretrage, već se heuristička vrednost svakog elementa inkrementalno gradi. Ovaj vid optimizacije omogućen je samo u slučaju pohlepnih heurističkih funkcija koje u svakom koraku algoritma u odnosu na redukovan skup $\mathcal{L}$ ocenjuju novoizabrano slovo. Nakon pro-

širivanja parcijalne reči odovarajućim slovom azbuke, vrši se ažuriranje pozicija na taj način što se inkrementiraju pozicije onih reči koje počinju izabranim slovom.

Najpre se vrši inicijalizacioni korak algoritma izvan glavne petlje. Funkcija **H** predstavlja heurističku funkciju kojoj se inicijalno prosleđuju pozicije na početak reči iz $\mathcal{L}$. Promenjiva težine sadrži slova i odgovarajuće težine svakog slova dobijene heurističkom ocenom. Zatim se popunjava bim elementima, gde svaki element sadrži slovo azbuke, odgovarajuću težinu i inicijalne pozicije. Nakon toga vrši se redukcija bima kao što je prikazano pod Algoritam 5. Ako je veličina bima manja od dozvoljene širine bima β vrši se ažuriranje pozicija svakog elementa kao što je prikazano pod Algoritam 6 i vrši se provera da li postoji element koji predstavlja rešenje PNZN na isti način koji je opisan pod Algoritam 2. Ako takav element postoji, postavlja se indikator za kraj algoritma, a pronađena parcijalna reč se čuva kao rešenje. Ažuriranjem pozicija inkrementiraju se pozicije koje odgovaraj rečima iz $\mathcal{L}$ počev od trenutnih pozicija, a koje počinju poslednje dodatim slovom. U slučaju da je veličina bima veća od dozvoljene širine elementi bima se najpre promešaju, a zatim sortiraju u odnosu na heurističku ocenu. Mešanjem elemenata se eliminiše konstantni azbučni poredak kada neki elementi imaju jednaku heurističku vrednost, a time i odabir takvih elemenata u azbučnom poretku. Zatim se vrši odabir β elemenata sa najvećom heurističkom ocenom, pritom se vrši ažuriranje pozicija odabranih elemenata i već opisana provera zadovoljenja rešenja PNZN. Glavna petlja algoritma prolazi kroz elemente bima, i za svaki element se računaju heurističke ocene slova azbuke u odnosu na trenutne pozicije tog elementa u $\mathcal{L}$. Od svakog elementa dobija se $|\Sigma|$ novih elemenata proširivanjem parcijalne reči elementa slovima azbuke. Na heurističku vrednost novog elementa dodaje se heuristička ocena odgovarajućeg slova. Novi element zadržava iste pozicije, koje se ažuriraju pri redukciji bima ali samo za odabrane elemente, čime se izbegava nepotrebno ažuriranje pozicija elemenata koji neće biti izabrani. Novi elementi se dodaju u bim, nakon čega se vrši redukcija bima na prethodno opisan način. Algoritam se završava kada pronađe jedno rešenje ili ako dubina dostigne $maxD$ jer u tom slučaju kao što je već pokazano postoji jedno trivijalno rešenje problema.

Iako algoritam pretrage bima ne garantuje pronalaženje optimalnog rešenja, ovako

Algoritam 4 PretragaBimaPNZN()

```
1: b1 ← {}
2: b2 ← {}
3: pronađenaNadniska ← Netačno
4:
5: težine ← H([0,...,0])
6:
7: for α ∈ Σ do
8:     težinaSlova ← 0
9:     for t ∈ težine do
10:        if α == t.slovo then
11:            težinaSlova ← t.vrednost
12:        end if
13:    end for
14:    element ← {α, težinaSlova, [0,...,0]}
15:    b1.dodaj(element)
16: end for
17:
18: b1 ← RedukujBim(b1, pronađenaNadniska)
19: dubina ← 0
20:
21: while ¬pronađenaNadniska ∧ dubina < maxD do
22:    for s ∈ b1 do
23:        težine ← H(s.pozicije)
24:        for α ∈ Σ do
25:            težinaSlova ← 0
26:            for t ∈ težine do
27:                if α == t.slovo then
28:                    težinaSlova ← t.vrednost
29:                end if
30:            end for
31:            element ← {(s.reč)α, s.hvrednost + težinaSlova, s.pozicije}
32:            b2.dodaj(element)
33:        end for
34:    end for
35:
36:    b1.obriši()
37:    b1 ← RedukujBim(b2, pronađenaNadniska)
38:    b2.obriši()
39:    dubina ← dubina + 1
40: end while
```

Algoritam 5 RedukujBim($\mathcal{B}$, *pronađenaNadniska*)

```
1: if B.veličina() ⩽ β then
2:     for e ∈ B do
3:         AžurirajPozicije(e)
4:         if ZajedničkaNadniska(e.reč) then
5:             nn ← s.reč                          ▷ nn - najbolja nadniska
6:             nd ← |nn|                           ▷ nd - najbolja dužina
7:             pronađenaNadniska ← Tačno
8:         end if
9:     end for
10:    return B
11: end if
12:
13: Promešaj(B)
14: Sortiraj(B)
15: Bp ← {}
16:
17: for i = 0 → β do
18:     AžurirajPozicije(B[i])
19:     if ZajedničkaNadniska(B[i].reč) then
20:         nn ← B[i].reč
21:         nd ← |nn|
22:         pronađenaNadniska ← Tačno
23:     end if
24:     Bp.dodaj(B[i])
25: end for
26:
27: B.obriši()
28: return Bp
```

konstruisan algoritam ima polinomijalno vreme izvršavanja $O(\beta * maxD)$ što ga u praksi čini daleko efikasnijim od prethodno opisanog algoritma granja sa odsecanjem. Iz tog razloga će ovaj algoritam biti testiran na većim instancama PNZN, gde je algoritam grananja sa odsecanjem praktično neizvodljivo testirati zbog eksponencijalne složenosti.

Algoritam 6 AžurirajPozicije(*element*)

```
1: poslednjeDodatoSlovo ← element.reč[element.reč.dužina() − 1]
2:
3: for i = 0 → ℒ.veličina() do
4:     if ℒ[i][element.pozicije[i]] == poslednjeDodatoSlovo then
5:         element.pozicije[i] ← element.pozicije[i] + 1
6:     end if
7: end for
```

Heurističke funkcije Većinsko Spajanje i Težinsko Većinsko Spajanje

Većinsko Spajanje predstavlja jedan od najpopularnijih algoritama koji rešava PNZN. To je pohlepni algoritam koji inkrementalno konstruiše nadnisku. Najpre se odredi slovo koje se najčešće nalazi na početku reči iz skupa $\mathcal{L}$, a zatim se izabrano slovo briše sa početka reči iz $\mathcal{L}$ koje ga sadrže. Postupak se ponavlja dok skup $\mathcal{L}$ ne postane prazan, a dobijena nadniska predstavlja rešenje PNZN [1]. U kontekstu opisanog algoritma pretrage bima, većinsko spajanje je korišćeno kao heuristička funkcija koja za dati niz pozicija u $\mathcal{L}$ vraća objekat koji sadrži slova azbuke i odgovarajuće težine u udnosu na to koliko reči od odgovarajućih pozicija počinje odgovarajućim slovom. Nakon izračunavanja težina, vrši se mešanje i sortiranje vrednosti. Mešanje nije neophodno, ali kao što je već rečeno eliminiše se konstantan azbučni poredak.

Algoritam 7 VećinskoSpajanje(pozicije)

```
1: težine ← {α ∈ Σ, [0, ..., 0]}                    ▷ inicijalizacija težina
2:
3: for i = 0 → ℒ.veličina() do
4:     if pozicije[i] ≤ (ℒ[i].dužina() − 1) then
5:         for t ∈ težine do
6:             if t.slovo == ℒ[i][pozicije[i]] then
7:                 t.vrednost ← t.vrednost + 1
8:             end if
9:         end for
10:    end if
11: end for
12:
13: Promešaj(težine)
14: Sortiraj(težine)
15:
16: return težine
```

Pseudo kod algoritma prikazan je pod Algoritam 7. Mana Većinskog Spajanja je to što ne može da prepozna globalnu strukturu reči iz skupa $\mathcal{L}$. U principu Većinsko spajanje izostavlja činjenicu da reči mogu biti različitih dužina. To dalje znači da će slova sa početka kraćih reči imati veću šansu da budu uklonjena iako algoritam i dalje treba da obradi preostale dugačke reči. Iz tog razloga bi skidanje slova sa početka kraćih reči trebalo da bude manje prioritetno. Drugim rečima bolje je da se prioritizira skidanje slova sa početka dužih reči. To se može postići dodavanjem težine svakom slovu azbuke u odnosu na dužinu preostalih reči iz $\mathcal{L}$ nakon uklanjanja tog simbola sa početka reči koje počinju tim simbolom. Dakle korak 7 u algoritmu VećinskoSpajanje možemo zameniti sa:

$$t.vrednost \leftarrow t.vrednost + (\mathcal{L}[i].dužina() - 1) - pozicije[i] \tag{2.4}$$

Ovako modifikovan algoritam naziva se Težinsko Većinsko Spajanje i postoje indikacije da na određenim instancama problema može da nadmaši algoritam Većinskog Spajana, pogotovo kada nema struktuiranosti unutar skupa $\mathcal{L}$ ili kada je ta struktuiranost haotična [1]. Predstavljene dve heurističke funkcije su korišćene u algoritmu pretrage bima.

Glava 3

Evaluacija algoritama

U ovom poglavlju biće prikazani rezultati izvršavanja predstavljenih algoritama kao i test podaci koji su korišćeni za testiranje i način na koji su test podaci generisani. Takođe će biti dat kratak pregled eksperimentalnog okruženja u kome se vršilo testiranje, hardverska i softverska specifikacija računara kao i specifičnosti implementacije.

3.1 Test podaci

Za uspešno testiranje rada predstavljenih algoritama pored instanci problema najkraće zajedničke nadniske $IP = \{\mathcal{I}_1(\Sigma_1, \mathcal{L}_1), ..., \mathcal{I}_k(\Sigma_k, \mathcal{L}_k)\}$ neophodno je da postoji barem jedno optimalno rešenje $\mathcal{R}_i$ za svaku instancu problema $\mathcal{I}_i$. Takvo rešenje najčešće nije jedinstveno, a broj optimalnih rešenja varira od instance do instance problema i raste sa porastom broja reči u skupu $\mathcal{L}$. Ono što je bitno, jeste dužina takvog rešenja i to je jedini validan parametar optimalnosti. Algoritam pretrage bima ne garantuje pronalaženje optimalnog rešenja dok algoritam grananja sa odsecanjem garantuje. Naizgled on može poslužiti za testiranje algoritma pretrage bima, ali vreme izvšavanja grananja sa odsecanjem eksponencijalno raste sa porastom parametra $maxD$ pa vreme izvšavanja ovog algoritma čak i na manjim instancama PNZN postaje predugo. Na primer ako najduža reč u skupu $\mathcal{L}$ ima dvadeset slova i pritom je data proizvoljna azbuka kardinalnosti $|\Sigma| = 4$, u najgorem slučaju algoritam grananja sa odsecanjem treba da obradi $(4^{80} - 1)/3$ čvorova. S obzirom na navedene činjenice i manjak test podataka u literaturi, osmišljen je algoritam koji generiše test instance. Pseudo kod algoritma je prikazan pod Algoritam 8. Ideja je da se odabere reč koja će predstavljati gornju granicu optimalnosti i da

se na osnovu odabrane reči generišu instance problema. Generator test instanci ima četiri parametra: *putanja* predstavlja putanju do tekstualnog fajla koji sadrži reč koja predstavlja gornju granicu, n označava broj slova u azbuci Σ, m označava broj reči u skupu $\mathcal{L}$, a parametar γ predstavlja verovatnoću uklanjanja slova iz gg pri generisanju reči. Najpre se učitava reč gg sa odabrane lokacije i azbuka popunjava odabranim slovima. Zatim se popunjava skup $\mathcal{L}$. U svakoj iteraciji petlje se kreće od prazne reči ω, prolazi se kroz slova gg i bira se slučajan broj iz uniformne raspodele (u granicama [0,1]). Ako je odabrani slučajan broj veći od γ, reč ω se proširuje trenutnim slovom. Zapravo, reč ω predstavlja reč koja se dobija uklanjanjem slova iz reči gg sa verovatnoćom γ. Na ovaj način se generiše m reči kojima se popunjava skup $\mathcal{L}$, a algoritam na kraju vraća jednu generisanu instancu problema $\mathcal{I}(\Sigma, \mathcal{L})$.

Algoritam 8 GeneratorTestInstanci($putanja, n, m, \gamma$)

```
1:  gg ← Pročitaj(putanja)                          ▷ gg - gornja granica
2:  Σ ← {}
3:  L ← {}
4:
5:  for i → n do
6:      α ← PročitajSlovo()
7:      Σ.dodaj(α)
8:  end for
9:
10: for i → m do
11:     ω ← ε
12:     for s ∈ gg do
13:         slučajanBroj ← GenerišiSlučajanBroj()
14:         if slučajanBroj > γ then
15:             ω.dodaj(s)
16:         end if
17:     end for
18:
19:     L.dodaj(ω)
20:
21: end for
22:
23: I ← (Σ, L)
24:
25: return I
```

Preostalo je još objasniti na koji način se generiše reč koja predstavlja gornju grani-

cu. Reč gg dobija se slučajnim odabirom slova iz azbuke. U svakoj iteraciji izgradnje bira se slučajan broj sb iz uniformne raspodele (u granicama $[0,|\Sigma|-1]$) i parcijalnoj reči dodaje odgovarajuće slovo azbuke $\Sigma[sb]$. Gornja granica optimalnosti garantuje jedno rešenje problema. Uz pažljivo odabran parametar γ i dovoljno veliki parametar m gornja granica će skoro uvek i predstavljati optimalno rešenje problema. U svakom slučaju gornja granica će u kontekstu testiranja predstavljenih algoritama biti sinonim za optimalno rešenje i poređenje rezultata će se vršiti u odnosu na njenu dužinu.

Sada će biti definisan skup instanci problema IP koji je korišćen za testiranje u sekciji 3.3. Postoje četiri bitna parametra koja su korišćena za generisanje instanci PNZN:

1. Veličina azbuke Σ ($n = |\Sigma|$)
2. Broj reči u skupu $\mathcal{L}$ ($m = |\mathcal{L}|$)
3. Dužina gornje granice ($|gg|$)
4. Verovatnoća uklanjanja slova (γ)

Veličina azbuke uzeta je iz opsega $n \in \{2, 4, 16\}$, broj reči u skupu $\mathcal{L}$ iz opsega $m \in \{10, 20, 40, 80\}$. Dužina gornje granice definiše i potencijalnu gornju granicu za dužinu reči u skupu $\mathcal{L}$ i ona je uzeta iz opsega $|gg| \in \{50, 100, 500, 2000\}$. Na kraju verovatnoća uklanjanja slova uzeta je iz opsega $\gamma \in \{0.1, 0.2\}$. Instance problema podeljene su u dva skupa $IP_{\gamma=0.1}$ i $IP_{\gamma=0.2}$ na osnovu verovatnoće uklanjanja slova. Oba skupa sadrže po 48 instanci problema koje su dobijene kao dekartov proizvod vrednosti prethodno opisanih parametara. Vrednosti širine bima uzete su iz opsega $\beta \in \{100, 200, 400\}$. Algoritam grananja sa odsecanjem testiran je na malom skupu instanci IPG, sa verovatnoćom uklanjanja $\gamma = 0.2$, $n \in \{2, 4\}$, $m \in \{8, 16, 32\}$ i $|gg| \in \{20, 24, 28\}$.

3.2 Eksperimentalno okruženje

U ovoj sekciji će biti dat kratak pregled hardverske i softverske konfiguracije računara na kome se vršilo testiranje kao i specifičnosti implementacije predstavljenih algoritama.

Hardversko i softversko okruženje

Svi rezultati izvršavanja algoritama dobijeni su na računaru pod Linux operativnim sistemom sledećih sistemskih specifikacija:

- Operativni sistem - Ubuntu 20.04.6 LTS
- Kernel - Linux 5.15.0-46-generic

Hardverske komponente računara na kome se vršilo testiranje su sledeće:

- Procesor - AMD Ryzen 3 1300X, 3.5GHz
- Arhitektura - x86-64
- Broj jezgara - 4
- Keš memorija - L1(384KiB), L2(2MiB), L3(8MiB)
- Ram memorija - Kingston, 8GB DDR4
- Grafička kartica - Asus Radeon RX 570 (4GB GDDR5)

Treba napomenuti još to da iako procesor ima četiri jezgra i četiri fizičke niti, algoritmi nisu implementirani tako da podržavaju paralelno izvršavanje.

Specifičnosti implementacije i pokretanje programa

Svi predstavljeni algoritmi u radu implementirani su u programskom jeziku C++ kako bi se postigla što veća efikasnost u evaluaciji. Korišćen je GCC 9.4.0 prevodilac i standard jezika C++20. Ceo programski kod je organizovan unutar klase, a svi prethodno navedeni algoritmi i pomoćne funkcije kao privatne funckije članice (*eng.* member function) kako bi podrazumevano bile utisnute (*eng.* inline) na mestima u programu na kojima se pozivaju i kako bi se time izbegli nepotrebni troškovi poziva. U cilju što veće optimizacije određenih delova programskog koda, iako u današnje vreme ne popularne, korišćene su i pokazivačke promenjive i odgovarajuća aritmetika. Implementacija svih struktura podataka poput skupova, vektora, uređenih parova uzeta je iz standardne biblioteke C++ jezika (*eng.* Standard C++ Library), kao i svi pomoćni algoritmi poput sortiranja ili mešanja elemenata. Svi generatori slučajnih brojeva koji su korišćeni sadrže fiksiranu vrednost semena (*eng.* seed), kako bi generisanje test instanci i evaluacija algoritama mogla da se reprodukuje u

definisanim uslovima. Sve generisane test instance i odgovarajući rezultati nalaze se u numerisanim tekstualnim fajlovima. Programski kod je organizovan u dva cpp fajla i prevodi se korišćenjem pomoćnog alata Make. Prilikom prevođenja upaljen je najviši nivo optimizacije od strane GCC prevodioca postavljanjem parametra -O3. Prevođenje i pokretanje odgovarajućih programa vrši se iz Linux terminala na sledeći način:

- make testInstanceGenerator - prevodi generator test instanci
- make SCS - prevodi program koji pokreće opisane algoritme
- ./testInstanceGenerator (režim rada) - pokreće generator test instanci koji nakon poziva očekuje putanju do tekstualnog fajla u kome se nalazi reč koja predstavlja gornju granicu, broj reči m, i željena slova azbuke (parametar γ se postavlja u kodu pre prevođenja), izlaz je numerisan tekstualni fajl koji predstavlja jednu instancu PNZN (postoji više režima rada generatora, na primer prethodno opisani podrazumeva ulazni argument 2, dok je za generisanje reči gg potrebno navesti ulazni argument 3)
- ./SCS (broj test instance) - pokreće program koji izvršava algoritme, kao ulazni argument očekuje redni broj test instance, izlaz je numerisan tekstualni fajl koji sadrži rezultat izvršavanja ulazne test instance

3.3 Eksperimentalni rezultati

U ovoj sekciji biće prikazani eksperimentalni rezultati izvršavanja algoritama. U tabeli 3.1 prikazani su rezultati algoritma grananja sa odsecanjem, na malom skupu test instanci $IPG_{\gamma=0.2}$. Kolone redom označavaju:

1. Veličinu ulazne azbuke Σ ($n = |\Sigma|$)
2. Broj reči u skupu $\mathcal{L}$ ($m = |\mathcal{L}|$)
3. Dužinu gornje granice ($|gg|$)
4. Dužinu nadniske koju je algoritam granja sa odsecanjem pronasao (GSO)
5. Vreme izvršavanja algoritma (t)
6. Minimalnu dubinu pretrage (minD)

7. Maksimalnu dubinu pretrage (maxD)

$IPG_{\gamma=0.2}$						
$\|\Sigma\|$	m	$\|gg\|$	GSO	$t(s)$	$minD$	$maxD$
2	8	20	20	1	19	38
2	8	24	24	6	21	42
2	8	28	28	101	24	48
2	16	20	20	0	19	38
2	16	24	24	6	21	42
2	16	28	28	105	26	52
2	32	20	20	0	19	38
2	32	24	24	6	22	44
2	32	28	28	108	26	52
4	8	12	12	4	11	44
4	8	13	13	15	13	52
4	8	14	14	64	14	56
4	16	12	12	4	11	44
4	16	13	13	15	13	52
4	16	14	14	63	14	56
4	32	12	12	4	12	48
4	32	13	13	15	13	52
4	32	14	14	64	14	56

Tabela 3.1: Rezultati grananja sa odsecanjem na test instancama $IPG_{\gamma=0.2}$

S obzrom na dužinu izvršavanja algoritma na instancama $m = \{16, 32\} \times |gg| = \{20, 24, 28\}$ rezultati nisu prikazani već je tabela skraćena. Vreme izvršavanja je duže od 3 sata.

$IP_{\gamma=0.1}$						
$\|\Sigma\|$	m	$\|gg\|$	VS	$t(s)$	TVS	$t(s)$
2	8	20	20	1	19	38
2	8	24	24	6	21	42
2	8	28	28	101	24	48
2	16	20	20	0	19	38
2	16	24	24	6	21	42
2	16	28	28	105	26	52
2	32	20	20	0	19	38
2	32	24	24	6	22	44
2	32	28	28	108	26	52
4	8	12	12	4	11	44
4	8	13	13	15	13	52
4	8	14	14	64	14	56

Tabela 3.2: Rezultati grananja sa odsecanjem na test instancama $IPG_{\gamma=0.2}$

Bibliografija

[1] Antonio J. Fernandez Christian Blum, Carlos Cotta and Francisco Gallardo. A Probabilistic Beam Search Approach to the Shortest Common Supersequence Problem. In *Lecture Notes in Computer Science*, 2007.

[2] Joseph Kruskal David Sankoff. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Center for the Study of Language and Inf, 1983.

[3] Garey and Johnson. Shortest common supersequence. on-line at: `https://www.csc.kth.se/~viggo/wwwcompendium/node165.html`.

[4] Storer JA. *Data compression: methods and theory.* Computer Science Press, 1988.

[5] Marc Huber Gunther Raidl Jonas Mayerhofer, Markus Kirchweger. A Beam Search for the Shortest Common Supersequence Problem Guided by an Approximate Expected Length Calculation. 2022.

[6] Frerk Schneider Jurgen Branke, Martin Middendorf. Improved heuristics and a genetic algorithm for finding short supersequences, 1998.

[7] Hon Wai Leong Kang Ning. Towards a better solution to the shortest common supersequence problem: the deposition and reduction algorithm, 2006.

[8] Esko Ukkonen Kari-Jouko Raiha. The shortest common supersequence problem over binary alphabet is NP-complete. In *Theoretical Computer Science*, 1981. Volume 16, Issue 2, Pages 187-198.

[9] David Maier. The Complexity of Some Problems on Subsequences and Supersequences, 1978.

[10] Gianluca Della Vedova Giancarlo Mauri Paolo Barone, Paola Bonizzoni. An Approximation Algorithm for the Shortest Common Supersequence Problem: An Experimental Analysis, 2001.

[11] Matt Payne. What is Beam Search? Explaining The Beam Search Algorithm. on-line at: `https://www.width.ai/post/what-is-beam-search`.

[12] Martin Middendorf Rene Michel. An island model based ant system with lookahead for the shortest supersequence problem. 2006.

[13] Timos K. Sellis. Multiple-query optimization, 1988. ACM Transactions on Database Systems (TODS), 13(1):23-52.

[14] Ming Li Tao Jiang. On the Approximation of Shortest Common Supersequences and Longest Common Subsequences, 1995.

[15] Ronald L. Rivest Clifford Stein Thomas H. Cormen, Charles E. Leiserson. *Introduction to Algorithms, Second edition.* MIT Press and McGraw-Hill, 2001.

[16] V. G. Timkovskii. Complexity of common subsequence and supersequence problems and related problems, 1989.

[17] Vadim G. Timkovsky. Some Approximations for Shortest Common Nonsubsequences and Supersequences. In *String Processing and Information Retrieval*, 2006.

Biografija autora

Vuk Stefanović Karadžić (*Tršić, 26. oktobar/6. novembar 1787. — Beč, 7. februar 1864.*) bio je srpski filolog, reformator srpskog jezika, sakupljač narodnih umotvorina i pisac prvog rečnika srpskog jezika. Vuk je najznačajnija ličnost srpske književnosti prve polovine XIX veka. Stekao je i nekoliko počasnih mastera. Učestvovao je u Prvom srpskom ustanku kao pisar i činovnik u Negotinskoj krajini, a nakon sloma ustanka preselio se u Beč, 1813. godine. Tu je upoznao Jerneja Kopitara, cenzora slovenskih knjiga, na čiji je podsticaj krenuo u prikupljanje srpskih narodnih pesama, reformu ćirilice i borbu za uvođenje narodnog jezika u srpsku književnost. Vukovim reformama u srpski jezik je uveden fonetski pravopis, a srpski jezik je potisnuo slavenosrpski jezik koji je u to vreme bio jezik obrazovanih ljudi. Tako se kao najvažnije godine Vukove reforme ističu 1818., 1836., 1839., 1847. i 1852.