# Algoritmi za poravnavanje bioloških nizova s primjenom na proteine virusa

**Tripić, Anabela**

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Anabela Tripić

# ALGORITMI ZA PORAVNANJE BIOLOŠKIH NIZOVA S PRIMJENOM NA PROTEINE VIRUSA

Diplomski rad

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

INTEGRIRANI PREDDIPLOMSKI I DIPLOMSKI SVEUČILIŠNI STUDIJ
FIZIKA I INFORMATIKA; SMJER NASTAVNIČKI

**Anabela Tripić**

Diplomski rad

# ALGORITMI ZA PORAVNANJE BIOLOŠKIH NIZOVA S PRIMJENOM NA PROTEINE VIRUSA

Voditelj diplomskog rada: doc. dr. sc. Andrej Novak

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____

2. _____

3. _____

Datum polaganja: _____

Zagreb, 2023.

Zahvaljujem se doc. dr. sc. Andreju Novaku na pruženom znanju, pomoći, savjetima i podršci kako tijekom izrade ovog diplomskog rada, tako i tijekom studiranja. Hvala mojoj obitelji, posebno mojim sekama na ogromnoj podršci i razumijevanju, što su me uvijek gurali naprijed. Hvala mome suprugu za bezuvjetnu podršku tijekom cijelog školovanja. Hvala svim prijateljima i kolegama što su moje studentske dane učinili boljima.
Hvala mome anđelu, mojoj baki na svemu.

# Sažetak

U bioinformatici, poravnanje nizova (ili sekvenci) je način raspoređivanja primarnih bioloških nizova DNA, RNA ili proteina kako bi se odredile regije sličnosti koje mogu biti posljedica funkcionalnih, strukturnih ili evolucijskih odnosa između nizova. Ako dva niza u poravnanju dijele zajedničkog pretka, nepodudarnosti se mogu tumačiti kao točkaste mutacije, a praznine kao mutacije umetanja ili brisanja uvedene u jednu ili obje loze u vremenu kad su se razišle. U poravnavanju proteinskih bioloških nizova, stupanj sličnosti između aminokiselina koje zauzimaju određenu poziciju u nizu može se interpretirati kao gruba mjera koliko je određena regija ili motiv biološkog niza očuvan među lozama. Odsutnost supstitucija, ili prisutnost samo vrlo konzervativnih supstitucija (supstitucija aminokiselina čiji bočni lanci imaju slična biokemijska svojstva) u određenoj regiji biološkog niza, sugeriraju da ova regija ima strukturnu ili funkcionalnu važnost. Iako su nukleotidne baze DNK i RNK sličnije jedna drugoj nego aminokiselinama, očuvanje uparivanja baza može ukazivati na sličnu funkcionalnu ili strukturnu ulogu. Vrlo kratki ili vrlo slični biološki nizovi mogu se poravnati ručno; međutim, najzanimljiviji problemi zahtijevaju poravnavanja dugih, vrlo varijabilnih ili iznimno brojnih bioloških nizova koji se ne mogu uskladiti isključivo ljudskim naporom. Računalni pristupi poravnavanju nizova općenito spadaju u dvije kategorije: globalna poravnanja i lokalna poravnanja. Određivanje globalnog poravnanja oblik je globalne optimizacije koja vodi poravnanje tako da se proteže cijelom dužinom svih nizova upita. Suprotno tome, lokalna poravnanja određuju regije sličnosti unutar dugih bioloških nizova koji se općenito jako razlikuju. Lokalna poravnanja često su poželjnija, ali ih može biti teže izračunati zbog dodatnog izazova identificiranja regija sličnosti. Cilj ovog diplomskog rada je opisati, analizirati i primijeniti računalne algoritme za problem poravnanja bioloških nizova, uključujući spore, ali formalno optimizirajuće metode poput dinamičkog programiranja i učinkovite heurističke algoritme za pretraživanje baze podataka. Praktični dio rada uključuje istraživanje sličnosti između pojedinih varijanti viralnih proteina, npr. protein šiljka (*engl. spike protein*) u varijantama SARS-CoV-2 virusa.

Ključne riječi: poravnanje bioloških nizova, globalno poravnanje, lokalno poravnanje, mjera sličnosti.

## Abstract

In bioinformatics, array alignment (or sequences) is a way of arranging primary DNA, RNA, or protein sequences to determine similarity regions that may be the result of functional, structural, or evolutionary relationships between sequences. If two strings in the alignment share a common ancestor, the mismatches can be interpreted as point mutations, and the gaps as mutations of insertion or erasure are introduced into one or both lineages at the time they diverged. In the alignment of protein sequences, the degree of similarity between amino acids that occupy a particular position in a sequence can be interpreted as a rough measure of how much a particular region or sequence motif is preserved among the lineages. The absence of substitutions, or the presence of only very conservative substitutions (amino acid substitution whose lateral chains have similar biochemical properties) in a particular sequence region, suggests that this region has structural or functional significance. Although nucleotide bases of DNA and RNA are more similar to each other than amino acids, preserving base pairing may indicate a similar functional or structural role. Very short or very similar sequences can be aligned by hand; however, the most interesting problems require alignments of long, highly variable, or extremely numerous sequences that cannot be reconciled solely by human effort. Computer methods to sequences alignment generally fall into two categories: global alignments and local alignments. Determining global alignment is a form of global optimization that guides alignment so that it extends along the entire length of all query strings. In contrast, local alignments determine similarity regions within long sequences that generally vary widely. Local settlements are often preferable but can be more difficult to calculate due to the additional challenge of identifying similar regions. The goal of this thesis is to describe, analyze, and apply computer algorithms for the problem of sequence alignment, including slow but formally optimizing methods such as dynamic programming and effective heuristic database search algorithms. The practical part of the paper includes research on the similarities between individual variants of viral proteins, eg. spike protein in variants of SARS-CoV-2 viruses.



Keywords: array alignment, global alignment, local alignment, region similarity.

# Sadržaj

# 1 Uvod

Bioinformatika je grana znanosti koja usko povezuje fiziku, biologiju i računarstvo, a ubrzano se razvijala zadnja četiri desetljeća. Sve veća dostupnost tehnologije i računalnih resursa rezultirala je stvaranjem velikih skupova bioloških podataka. Veličina i posebnosti tih podataka motivirale su razvoj novih računalnih metoda koje bi omogućile njihovu pohranu, obradu, analizu i prikaz. Tu se javlja potreba za spajanjem nekoliko znanstvenih disciplina kao što su molekularna biologija, fizika, računalne znanosti, a sve to s ciljem proučavanja virusnih, bakterijskih i eukariotskih genoma. Ovakve metode koriste se sve više u znanstvene svrhe, baš zbog toga što povezuju proučavanje gena s različitim tehnikama bioinformatike. Sve veća količina podataka koju znanstvenici imaju na raspolaganju omogućuje izradu statističkih i matematičkih modela na osnovu kojih se mogu odrediti potencijalne buduće pojave. Poravnanje bioloških nizova je najčešći prvi korak u bioinformatičkoj analizi, bilo da je u pitanju pronalazak evolucijski očuvanih regija među vrstama ili analiza genetske bolesti. Upravo zbog svoje široke primjene, poravnanje dva biološka slijeda predstavlja jedan od najstarijih i najviše istraživanih problema u bioinformatici. Cilj poravnanja je pronalazak sličnih ili identičnih regija u biološkim nizovima koje ukazuju na genetsku sličnost, svojevrsne mutacije koje su se dogodile ili pak mogu ukazivati na neke bitne funkcionalnosti koje su zastupljene u kodirajućim regijama genoma.

## 1.1 Biološke osnove

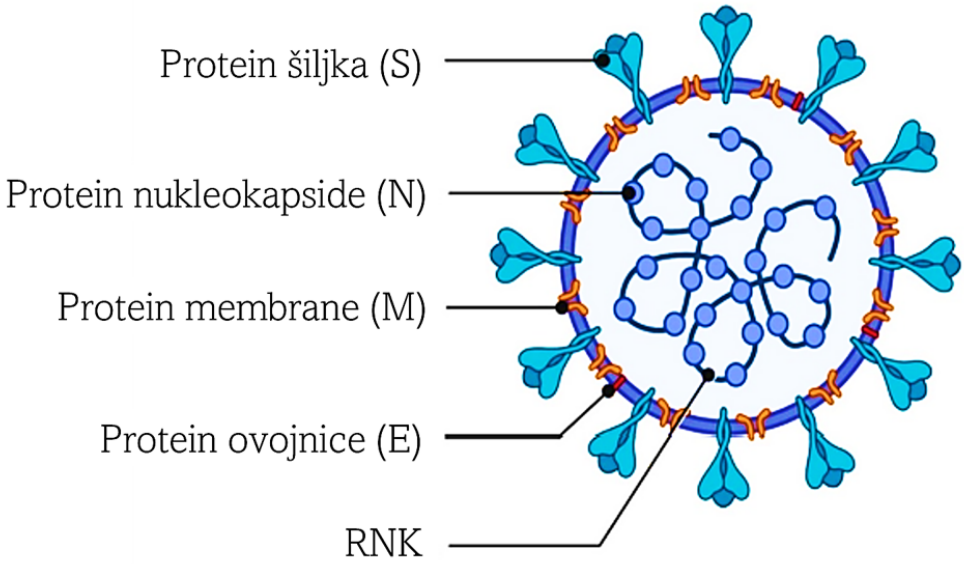
Za funkcioniranje svih poznatih živih organizama presudne su makromolekule nukleinskih kiselina, proteina i ugljikohidrata – takozvani biopolimeri. Dvije temeljne vrste nukleinskih kiselina su RNK i DNK. DNK služi kao spremište instrukcija za razvoj i funkcioniranje svih živih organizama, s izuzetkom RNK virusa. Postoji nekoliko vrsta RNK molekula koje dijelimo ovisno o njihovim funkcijama. Najvažnije su glasnička RNK (mRNK), transportna RNK (tRNK), ribosomska (rRNK) te regulacijske RNK poput mikro RNK (miRNK), male jezgrene RNK (snRNK) i male interferirajuće RNK (siRNK). Jednim imenom sve RNK koje možemo naći u stanici nazivamo transkriptomom. Osnovna jedinica nasljeđivanja u živim organizmima je gen. Gen je dio DNK ili RNK koji kodira informaciju za proizvodnju proteina ili RNK lanaca koji imaju aktivnu funkciju u organizmu. Većina DNK molekula su dvostruke uzvojnice koje se sastoje od manjih gradivih jedinica-nukleotida. Svaki lanac nukleotida se sastoji od nukleinskih baza (adenin, citozin, gvanin i timin) koje označavamo slovima A, C, G i T. Ta dva lanca su međusobno povezana tako da se adenin spaja s timinom, a gvanin s citozinom. Lanci se protežu u suprotnim smjerovima i zbog toga kažemo da su antiparalelni. RNK molekule imaju ulogu u kodiranju, dekodiranju, regulaciji i ekspresiji gena. Molekula RNK je poput DNK građena od nukleotida, ali za razliku od DNK je najčešće građena od jednog lanca. Nukleinske baze koje uz šećer i fosfatnu grupu tvore lanac RNK su adenin, citozin, gvanin i uracil koje označavamo sa slovima A, C, G i U. [1]

Slika 1.1: Grafički prikaz molekule RNK (lijevo) i DNK (desno).

Kodon je niz triju nukleotida koji se nalazi na molekuli glasničke RNK, odgovara određenoj aminokiselini ili stop signalu tijekom sinteze proteina. Određuje položaj određene aminokiseline u procesu sinteze bjelančevina u stanici, čime djeluju kao jedinica genetičkoga kodiranja. Svaka trojka nukleotida prepisana s DNK na glasničku RNK naziva se kodon. Nukleotidi su označeni slovima A, U, G i C. Svaka znamenka kodona može imati neku od $4$ vrijednosti, prema tome kodon DNK ima $64$ moguće vrijednosti, u usporedbi s binarnim bajtom koji ima $256$. Tipičan primjer DNK kodona je 'GCC', koji kodira aminokiselinu alanin. Veći broj ovih aminokiselina u kombinaciji naziva se polipeptid ili protein. Molekule DNK i RNK ispisane su jezikom od četiri nukleotida, dok jezik proteina uključuje $20$ aminokiselina. Kodoni daju ključ koji omogućuje međusobno prevođenje ova dva jezika. Svaki kodon odgovara jednoj aminokiselini (ili stop signalu), a cijeli skup kodona naziva se genetski kod. Genetski kod uključuje $64$ moguće permutacije ili kombinacije nukleotidnih nizova od tri slova koji se mogu načiniti od četiri nukleotida. Od $64$ kodona, $61$ predstavlja aminokiseline, a tri su stop signali. Na primjer, kodon 'CAG' predstavlja aminokiselinu glutamin, a 'TAA' je stop kodon. Jedna aminokiselina može biti kodirana s više od jednog kodona. Kada se kodoni čitaju iz nukleotidnog niza, oni se čitaju uzastopno i ne preklapaju se jedan s drugim. [3]

## 1.2 Protein šiljka

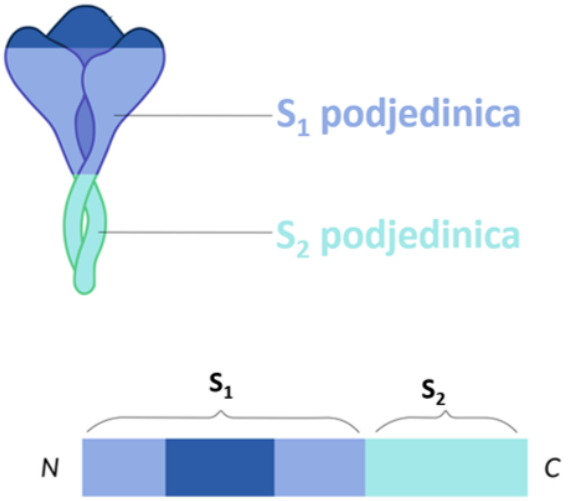
Protein šiljka (S protein/spike protein) veliki je transmembranski protein tipa I [1] u rasponu od 1160 do 1400 aminokiselina. Proteini u obliku šiljaka okupljaju se u skupine po tri člana na površini viriona kako bi formirali prepoznatljivu krunu (*lat. coronu)*.




Slika 1.2: Shematski prikaz virusa SARS-CoV-2 i njegovih strukturnih proteina. Označeni su protein šiljka (S), protein nukleokapside (N), protein membrane (M), protein ovojnice (E) i molekula RNK. Preuzeto s [4].

U usporedbi s M i E proteinima koji su primarno uključeni u sastavljanje virusa, protein šiljka igra ključnu ulogu u prodiranju u stanice domaćina i pokretanju infekcije. Naime, prisutnost proteina šiljka na koronavirusima je ono što dovodi do izbočina u obliku šiljaka na njihovoj površini. Raznolikost SARS-CoV virusa odražava se u varijabilnim proteinima šiljka koji su evoluirali u oblike koji se razlikuju u svojim receptorskim interakcijama i njihovom odgovoru na različite uvjete. Kod većine sojeva SARS-CoV-2 virusa, virioni sadrže protein šiljka koji nije podijeljen, dok se u nekim sojevima SARS-CoV-2 virusa protein šiljka nalazi podijeljen između $S_1$ i $S_2$ podjedinica. [4] Protein šiljka također je protein koji je odgovoran za dopuštanje virusu da se pričvrsti i stopi s membranom stanice domaćina. $S_1$ podjedinica je odgovorna za pričvršćivanje, dok je $S_2$ podjedinica odgovorna za strategije ulaska virusa u stanicu.

[1] Tip proteina koji se proteže preko cijele ćelije membrane. Funkcioniraju kao ulazi koji omogućuju transport specifičnih supstanci preko membrane.




Slika 1.3: Prikaz građe jednog šiljka koji se nalazi na površini proteina šiljka SARS-CoV-2 virusa. Ispod je prikaz podjele proteina šiljka na $S_1$ i $S_2$ podjedinicu.

Jedna od poznatijih značajki proteina šiljaka SARS-CoV-2 je mogućnost promijene kako se virus širi ili mijenja tijekom vremena tijekom evolucije virusa. Kodiran unutar virusnog genoma, protein može mutirati i mijenjati svoja biokemijska svojstva kako se virus razvija. Većina mutacija neće biti korisna ili će zaustaviti rad proteina šiljka ili pak neće utjecati na njegovu funkciju, ali neke mutacije mogu uzrokovati promjene koje novoj verziji virusa daju prednost čineći da se virus brže širi ili da je zarazniji. Jedan od načina na koji bi se to moglo dogoditi je kroz mutaciju na dijelu proteina šiljka koji sprječava zaštitna antitijela da se vežu na njega. Drugi način bio bi učiniti šiljke "ljepljivijima" za ljudske stanice. Zbog toga su nove mutacije koje mijenjaju strukturu i funkciju šiljaka potencijalno zabrinjavajuće–one mogu utjecati na kontrolu širenje SARS-CoV-2. [5]

## *1.3 Poravnanja*

Kada govorimo o poravnanju bioloških nizova, to je još uvijek aktualan problem koji nije u potpunosti riješen. Tu se radi se o usporedbi dvaju nizova čija se duljina mjeri u nekoliko stotina milijuna parova baza. Upravo zbog tako velike količine podataka potrebno je razviti algoritme i strukture podataka pomoću kojih će se moći obaviti poravnanje, koje za cilj ima pronalazak sličnih ili identičnih regija u genomima koje bi mogle upućivati na genetsku sličnost, svojevrsne mutacije koje su se dogodile ili pak ukazivati na neke bitne funkcionalnosti zastupljene u kodirajućim regijama genoma.

### 1.3.1 Globalno poravnanje

Algoritmi za globalno poravnanje pokušavaju poravnati svaki član svakog biološkog niza, odnosno osiguravaju da poravnanje sadrži sve članove iz niza $a$ i iz niza $b$. Algoritmi za globalno poravnanje najčešće se koriste kod poravnanja bioloških nizova koji su relativno slični te su otprilike jednake duljine.

### 1.3.2 Lokalno poravnanje

Lokalno poravnanje je vrsta poravnanja gdje se dijelovi upitnog niza pokušavaju na lokalno najbolji način poravnati s ciljanim nizom iz baze podataka. Kod takvih nizova poravnanje ne mora sadržavati sve članove iz niza $a$ i $b$. Ako biološki nizovi koji sudjeluju u poravnanju nisu homologni cijelom svojom dužinom, onda se mogu poravnati po dijelovima koji su ostali sačuvani (*engl. conserved regions*) i tada govorimo o lokalnom poravnanju (npr. poravnanje proteina koji imaju istu funkciju između različitih vrsta). [1]

### 1.3.3 Poluglobalno poravnanje

Poluglobalno poravnanje je varijanta globalnog poravnanja koja dopušta praznine na početku i/ili kraju jednog od bioloških nizova. Algoritmi za poluglobalno poravnanje pokušavaju pronaći najbolje moguće poravnanje koje sadrži sve članove niza $a$, a ne mora sadržavati sve članove niza $b$. Nastaju kao spoj algoritama za lokalno i algoritama za globalno poravnanje čineći njihovu kombinaciju. Jedan primjer njihovog korištenja je onaj u kojem je niz $a$ puno kraći od niza $b$, te poravnanjem želimo pronaći regiju niza $b$ koja je najsličnija nizu $a$. [7]

# 2 Algoritmi za poravnanja

## *2.1 Gap penalty-kažnjavanje praznina*

Gap penalty ili kažnjavanje praznina je metoda vrednovanja poravnanja dva ili više bioloških nizova. Prilikom poravnavanja nizova, uvođenje praznina može omogućiti algoritmu za poravnanje da pronalazi preklapanje s više pojmova nego što to može poravnanje bez praznina. Međutim, smanjenje praznina u poravnanju važno je za stvaranje korisnog poravnanja. Previše praznina može uzrokovati da poravnanje postane besmisleno. Kazne za praznine koriste se za prilagodbu rezultata poravnanja na temelju broja i duljine praznina. Praznine (*engl. gap*) su mjesta u nizu koja su popunjena uzastopnim znakom $-$. Prema tome, niz $AT{-}{-}CCTGA$ ima jednu prazninu duljine $3$, jer je sastavljena od tri znaka praznine, a niz $A{-}T{-}ACCT{-}A$ ima tri praznine duljine $1$, jer svaka praznina odgovara jednom znaku praznine. Nekoliko glavnih načina kažnjavanja praznina su konstantni, linearni, afini i konveksni model kažnjavanja praznina. [9]

### 2.1.1 Konstantno kažnjavanje

Najjednostavnija metoda kažnjavanja praznina u nizu je konstantno kažnjavanje. Ono je definirano tako da dodajemo konstantnu negativnu vrijednost $c$ cijeloj praznini, tako da svaka praznina u nizu sudjeluje s $-c$ bodova, neovisno koliko znakova praznina postoji.

### 2.1.2 Linearno kažnjavanje

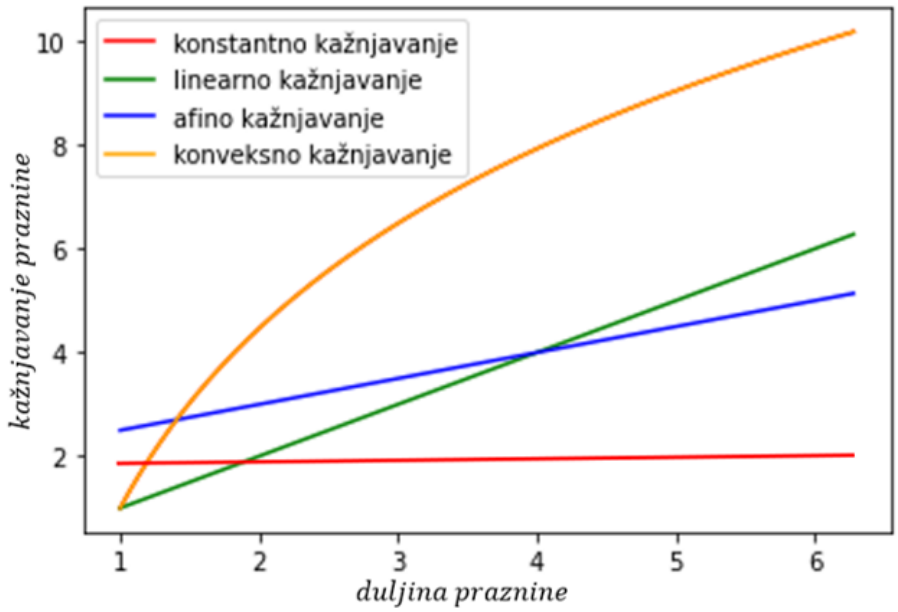
U usporedbi s konstantnom kaznom za prazninu, linearna kazna za prazninu uzima u obzir duljinu $L$ svakog umetanja ili brisanja u praznini. Dakle, ako je kazna za svaki umetnuti ili izbrisani element $X$ i duljina razmaka $L$ tada bi ukupna kazna praznine bila umnožak $LX$. Ova metoda daje prednost kraćim prazninama, s ukupnim rezultatom koji se smanjuje sa svakom dodatnom prazninom.

### 2.1.3 Afino kažnjavanje

Najčešće korišteno kažnjavanje praznina je afino kažnjavanje. Ono, za zadane konstantne vrijednosti $i$ i $j$, svaku prazninu kažnjava s $i + (jL)$, pri čemu je $L$ duljina praznine. U ovom slučaju se vrijednost $i$ naziva kazna započimanja praznine, a vrijednost $j$ kazna proširenja praznine. Kazna započimanja praznine $i$ se naplaćuje za pokretanje praznine svojim prvim simbolom, dok se kazna za proširenje praznine $j$ naplaćuje za svaki sljedeći simbol dodan u prazninu. Često je vrlo bitno pitanje odabira vrijednosti $i$ i $j$, jer njihov odabir značajno utječe na konačni rezultat. [10] Općenito pravilo kojim se treba voditi je da, u slučaju poravnavanja evolucijski bliskih nizova u kojima se ne preferira veliki broj praznina, poželjno je jače kazniti svaku prazninu, dok je kod poravnavanja evolucijski udaljenijih nizova poželjno smanjiti kažnjavanje praznina. Također ovisno o konkretnom problemu koji istražujemo potrebno je odlučiti preferira li se manje duljih ili više kraćih praznina i sukladno tome odabrati parametre $i$ i $j$. [11]

### 2.1.4 Konveksno kažnjavanje

Smatrajući da afino kažnjavanje ne modelira dovoljno dobro stvarnu situaciju iz prirode, razvijena je fleksibilnija metoda konveksnog kažnjavanja ili logaritamskog kažnjavanja. Korištenje kazne afinog razmaka zahtijeva dodjeljivanje fiksnih kaznenih vrijednosti i za otvaranje i za proširenje razmaka. Konveksno kažnjavanje ima oblik $i + jln(L)$ , gdje je $i$ kazna započimanja praznine, vrijednost $j$ kazna proširenja praznine i $L$ je duljina praznine. Logaritamska kazna razmaka je izumljena kako bi se modificiralo afino kažnjavanje, tako da su dugi razmaci poželjni. Međutim, utvrđeno je da je korištenje logaritamskih modela dalo loše rezultate poravnanja u usporedbi s afinim modelom kažnjavanja. Ova metoda je motivirana studijama koje su pokazale da distribucija duljine praznina u prirodi prati zakon potencija (duljina praznine je linearno povezana sa svojom učestalosti). [11]




Slika 2.1: Grafički prikaz usporedbe funkcija koje opisuju kažnjavanje praznina.

## 2.2 BLAST-Basic Local Alignment Search Tool

Alat za osnovno pretraživanje lokalnog poravnanja (BLAST) pronalazi područja lokalne sličnosti između bioloških nizova. Program koji uspoređuje nizove nukleotida ili proteina s bazama podataka bioloških nizova i računa statističku značajnost podudaranja. BLAST se može koristiti za istraživanje funkcionalnih i evolucijskih odnosa između bioloških nizova, kao i za pomoć u identificiranju članova obitelji gena. Prije nego što su razvijeni brzi algoritmi i različite aplikacije, pretraživanje baza podataka zahtijevalo je značajne računarske resurse jer je korištena potpuna procedura usklađivanja (npr. Smith–Watermanov algoritam-kasnije algoritam za lokalno poravnanje).

BLAST je proizašao iz stohastičkog modela iz 1990. Samuela Karlina i Stephena Altschula. Oni su predložili metodu za procjenu sličnosti između poznatog slijeda DNK jednog organizma s onim drugog, a njihov je rad opisan kao statistički temelj za BLAST. Nakon toga, Altschul, zajedno s Warrenom Gishom, Webbom Millerom, Eugeneom Myersom i Davidom J. Lipmanom iz Nacionalnog instituta za zdravlje dizajnirao je algoritam BLAST, koji je objavljen u Journal of Molecular Biology 1990. i citiran više od 75 000 puta. [18]

BLAST ne može garantirati optimalno usklađivanje upitnog niza i baze podataka bioloških nizova kao što to čini Smith-Watermanov algoritam. Optimalnost Smith-Watermana osigurala je najbolju izvedbu točnosti i najpreciznije rezultate na račun vremena i snage računala.

Za BLAST postoji upitni niz i ciljni niz ili baza podataka bioloških nizova. Upitni niz je niz za koji želimo saznati sličnost, a ciljni niz je niz ili baza podataka bioloških nizova prema kojoj je upitni niz poravnat. Blast vraća izlaz u obliku tablica pogodaka koje su raspoređene u opadajućem redoslijedu prema odgovarajućem pristupnom broju zajedno sa svojim naslovima, pokrivenošću upita, identitetom niza i rezultatom. BLAST ima različite programe za poravnavanje nizova nukleotida, proteina itd. Sastoji se od višestrukih drugih BLAST programa, ali osnovne vrste BLAST-a su sljedeće:

BLASTn je vrsta gdje je upitni niz nukleotid, a ciljni niz je također nukleotid BLASTp je protein-protein BLAST gdje je upitni niz protein dok je i ciljani niz također protein. BLASTx-u ovoj vrsti BLAST-a, upitni niz je niz nukleotida, a ciljani niz je niz ili baza podataka proteina. Prvo se nukleotidni niz pretvara u svoj proteinski niz u tri okvira čitanja, a zatim se traži prema proteinu.

U tBLASTnu, upit je protein, a cilj je nukleotidni niz ili baza podataka. Ovdje se niz proteina pretražuje prema bazi podataka nukleotida koja se prevodi u odgovarajuće proteine. Prijevod se događa u svim okvirima čitanja, ali se uspoređuju samo 3 okvira čitanja.

TBLASTx je vrsta BLAST-a u kojoj je se upitni niz nukleotida i ciljani niz se prevode u svoje odgovarajuće proteinske nizove i zatim se međusobno poravnavaju. I upit i ciljni niz prevode se u svih 6 okvira čitanja. [13]

## 2.3 Needleman-Wunsch algoritam

Algoritam za globalno poravnanje nizova koji se temelji se na dinamičkom programiranju. Dinamičko programiranje je pristup rješavanja složenijih problema svođenjem na manje i lakše potprobleme koji su međusobno povezani. Rezultati manjih potproblema računaju se samo jednom i spremaju u memoriju. Kada se sljedeći put pokaže potreba za izračunom nekog većeg potproblema koji u sebi sadrži manji potproblem, rješenje se samo pročita iz memorije i time se uštedi vrijeme koliko bi ponovno računanje trajalo. Da bi sve funkcioniralo kako treba, rješenja svih dosad izračunatih potproblema moraju biti ispravno pohranjena i indeksirana, kako bi se omogućilo njihovo jednostavno i brzo čitanje. Jedan od algoritama dinamičkog programiranja za poravnanje više nizova je proširenje algoritma za poravnanje dva niza kojeg su predložili Needleman i Wunsch 1970.

Needleman-Wunschov algoritam uspoređuje svaki znak iz jednog niza duljine $m$ sa svakim znakom iz drugog niza duljine $n$. Bodovanje poravnanja sprema se u dvodimenzionalno polje (matricu) dimenzija $mn$. Element matrice u donjem desnom kutu sadrži konačnu vrijednost poravnanja, a samo poravnanje određuje se kao put od elementa u donjem desnom kutu matrice prema elementu u gornjem lijevom kutu rekurzivno sljedeći put najboljeg rezultata. Optimalno poravnanje obično se nalazi oko dijagonale matrice. Započinjemo inicijalizacijom matrice s mogućim rezultatima na sljedeći način:

$$F(0,0) = 0, \tag{2.1}$$

$$F(i,0) = F(i-1,0) - d, \tag{2.2}$$

$$F(0,j) = F(0,j-1) - d, \tag{2.3}$$

gdje je $d$ iznos kažnjavanja praznine. Bitno je inicijalizirati početne retke i stupce tako da algoritam može sustavno popunjavati podatke u ostalim elementima matrice. Element matice $F(0,0)$ je inicijaliziran s $0$, jer još nisu napravljena nikakva poravnanja. Promjenu stanja u redovima matrice iz $F(i-1,0)$ u $F(i,0)$ interpretiramo tako da je biološki niz $b$ imao prazninu, također promjenu stanja u stupcima matrice iz $F(0,j-1)$ u $F(0,j)$ interpretiramo tako da je biološki niz $a$ imao prazninu.

Drugi korak je popuniti matricu $F$ optimalnim rezultatima. Na svakoj poziciji $F(i,j)$, postoje četiri mogućnosti koje treba razmotriti:

- Niz $a$ ima razmak u trenutnom poravnanju: $F(i, j-1)$.
- Niz $b$ ima razmak u trenutnom poravnanju: $F(i-1, j)$.
- Postoji zamjena nukleotida na trenutnoj poziciji: $F(i-1, j-1)$.
- Postoji podudaranje na trenutnoj poziciji: $F(i-1, j-1)$.

Svaki od ovih scenarija daje različite rezultate. Kako bismo maksimizirali naš konačni rezultat i odredili optimalno poravnanje, moramo uzeti maksimum iz sljedećih scenarija:

$$F(i,j) = max \begin{cases} F(i-1,j) + d, \\ F(i,j-1) + d, \\ F(i-1,j-1) + S(x_i, y_j). \end{cases} \tag{2.4}$$

Pri čemu je funkcija $S(x_i, y_i)$ definirana kao:

$$S(x_i, y_j) = \begin{cases} 1, \ x_i = y_j \\ -1, \ x_i \neq y_j. \end{cases} \tag{2.5}$$

Poravnanje nizova završava dolje desno odnosno u $F(m,n)$. To je zato što je na poziciji $F(m,n)$ završetak oba niza, pa nema daljnjih promjena. Važno je da se pri ispunjavanju matrice sjećamo puta kako smo tamo stigli (npr. koji je element matrice dao maksimalan rezultat). Nakon inicijalizacije matrice, posljednji korak je praćenje kroz matricu počevši od donjeg desnog kuta do gornjeg lijevog (ili od $F(m,n)$ do $F(0,0)$). Na ovaj način će se osigurati optimalno poravnanje. Usporedbe se rade od najmanje jedinice značaja, para aminokiselina, po jedne iz svakog niza proteina. Sve moguće usporedbe predstavljene su putovima kroz niz. Numerička vrijednost, u ovom slučaju jedna, dodjeljuje se svakom elementu matrice u nizu koji predstavlja slične aminokiseline. Maksimalno podudaranje je najveći broj koji bi proizašao iz zbrajanja vrijednosti elemenata matrice svakog puta. [8]

Da bismo odredili složenost ovog algoritma potrebno je posebno analizirati svaki njegov dio (prema koracima izvođenja). Za početak, inicijalizacija matrice je vremenske složenosti $O(m+n)$. Sljedeće je popunjavanje matrice $F(i,j)$ s rezultatima. Za određivanje rezultata svakog elementa matrice potrebno ga je usporediti sa susjedima

(lijevi, gornji, dijagonalni) i to se izvodi u konstantnom vremenu. Za popunjavanje cijele matrice potrebno je $O(mn)$. Konačno povratno praćenje zahtjeva određen broj koraka. Možemo koračati po najviše $n$ redaka i $m$ stupaca te nam to daje složenost $O(m+n)$. Sljedeće u povratnom praćenju je pronaći konačan put koji zadovoljava uvjete. Budući da taj korak uključuje maksimalno $n$ redaka (gdje je $n > m$) to zahtjeva $O(n)$. Ukupna vremenska složenost algoritma tada je

$$O(m+n) + O(mn) + O(m+n) + O(n) = O(mn). \tag{2.6}$$

## 2.4 Smith-Waterman algoritam

Smith-Watermanov algoritam koji se također temelji na paradigmi dinamičkog programiranja izvodi lokalno poravnanje bioloških nizova. Koristi se za određivanje sličnih regija između dva biološka niza nukleinskih kiselina ili niza proteina. Umjesto da gleda cijeli biološki niz, Smith–Watermanov algoritam uspoređuje segmente svih mogućih duljina i optimizira bodovanje. Kao i kod Needleman-Wunsch algoritma postoje tri osnovna koraka. Prvi korak je inicijalizacija, gdje se prvi red i prvi stupac inicijaliziraju na $0$.

$$F(i, 0) = 0, \tag{2.7}$$

$$F(0, j) = 0. \tag{2.8}$$

Sljedeći korak je popunjavanje matrice, gdje se vrijednost elementa matrice računa uzimanjem maksimalne od sljedeće tri vrijednosti koje tvore rekurzivnu relaciju. Prva je vrijednost zbroj neposrednog lijevog unosa i funkcije bodovanja praznine, zatim zbroj vrijednosti neposrednog gornjeg unosa i funkcije bodovanja praznine i na kraju zbroj vrijednosti neposrednog gornjeg lijevog dijagonalnog unosa i funkcije sličnosti.

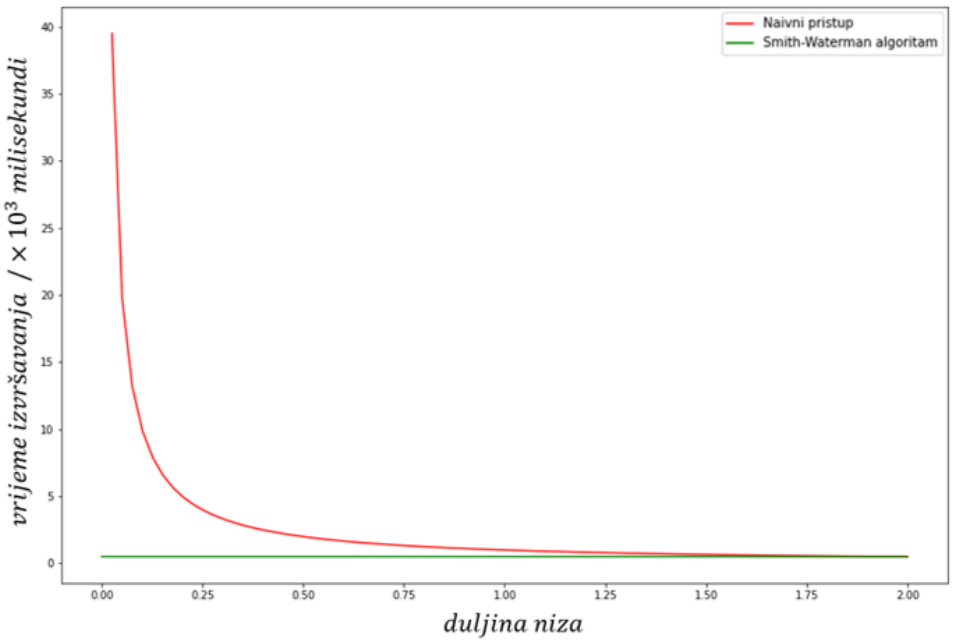
$$F(i, j) = max \begin{cases} 0, \\ F(i, j-1) + d, \\ F(i-1, j) + d, \\ F(i-1, j-1) + S(x_i, y_j). \end{cases} \tag{2.9}$$

Pri čemu je funkcija $S(x_i, y_j)$ definirana kao:

$$S(x_i, y_j) = \begin{cases} 1, \ x_i = y_j \\ -1, \ x_i \neq y_j. \end{cases} \tag{2.10}$$

Glavna razlika u odnosu na Needleman-Wunschov algoritam je da su pojedini elementi matrice s negativnim bodovanjem postavljeni na nulu, što lokalna poravnanja (dakle s pozitivnim bodovanjem) čini vidljivima. Treći korak je povratno praćenje (*engl. Trace-back*). Procedura povratnog praćenja počinje od elementa matrice s najvišim rezultatom i nastavlja se sve dok se ne naiđe na element matrice s rezultatom nula, čime se dobiva lokalno poravnanje s najvišim rezultatom. Pomaci prema dijagonali stvaraju podudaranje ili nepodudaranje, pomaci prema vrhu proizvode umetanje u očitanje, a pomaci u lijevo označavaju brisanje u očitanju. Algoritam za lokalno poravnanje pronalazi poravnanje samo na dijelovima nizova. Zbog svoje kvadratne složenosti u vremenu i prostoru, često se ne može praktično primijeniti na velike probleme i zamjenjuje se u korist manje općih, ali računalno učinkovitijih alternativa.

Što se tiče vremenske složenosti Smith-Waterman algoritma, vrlo je slična vremenu izvođenja Needleman-Wunsch algoritma. Neka su $n$ i $m$ duljine nizova koji sudjeluju u poravnanju. Ako promatramo naivan pristup (*engl. brut force*) tada bi vremenska složenost ovog algoritma bila $O(n^3m^3)$ jer podrazumijeva usporedbu svakog pojedinog mogućeg uređivanja svakog mogućeg podniza dvaju nizova međusobno, ali kada koristimo paradigmu dinamičkog programiranja odnosno Smith-Watermanov algoritam tada je vrijeme izvođenja $O(nm)$.




Slika 2.2: Grafički prikaz usporedbi vremena izvođenja programa naivnim pristupom i pristupom pomoću Smith-Waterman algoritma.

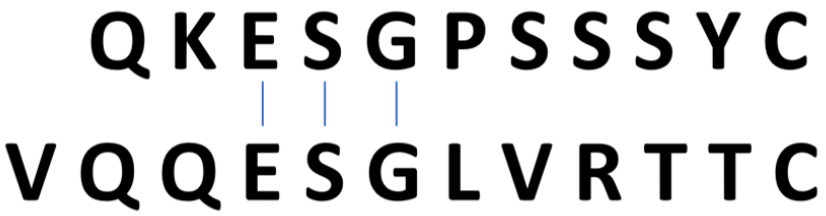
## 2.5 Usporedba globalnog i lokalnog poravnanja

| GLOBALNO PORAVNANJE | LOKALNO PORAVNANJE |
|---|---|
| Primjenjuje se na cijeloj duljini biološkog niza, od početka do kraja. | Pronalazi lokalne regije s najvećom mjerom sličnosti između dva biološka niza. |
| Ako dva biološka niza imaju približno jednaku duljinu primjerene su za globalno poravnanje. | Bilo koja dva biološka niza mogu se lokalno poravnavati jer se traže sličnosti na dijelovima bez obzira na ostatak niza. |
| Sadrži sva slova iz upita i ciljanog niza. | Sadrži podniz niza upita i podniza ciljanog niza. |
| Najčešće se koristi za usporedbu homolognih gena kao na primjer usporedba dvaju gena s istom funkcijom (usporedba gena čovjeka i miša) ili usporedba dvaju proteina sa sličnom funkcijom. | Koristi se za pronalaženje očuvanog uzorka u DNK, očuvanih domena ili motiva u dva proteina. |
| Najpoznatija tehnika globalnog poravnanja je Needleman-Wunsch algoritam. | Najpoznatija tehnika lokalnog poravnanja je Smith-Waterman algoritam |
| Neki od poznatijih alata: EMBOSS Needle, Needleman-Wunsch Global Align Nucleotid Sequences (Specijalni BLAST). | Neki od poznatijih alata: BLAST, EMBOSS Water, LALIGN. |

Tablica 2.1: Usporedba lokalnog i globalnog poravnanja.

Pogledajmo na istom primjeru kako izgleda lokalno, a kako globalno poravnanje. Neka je prvi biološki niz $s_1 = QKESGPSSSYC$ i neka je drugi niz $s_2 = VQQESGLVRTTC$


Slika 2.3: Prikaz globalnog poravnanja.




Slika 2.4: Prikaz lokalnog poravnanja.

Možemo primjetiti da je lokalno poravnanje neprekidno, što bismo mogli protumačiti da smo stigli do elementa matrice u kojem je vrijednost nula.

# 3 Primjena algoritma na proteinu šiljka

U ovom poglavlju primijenit ćemo gore opisani Needleman-Wunsch algoritam na biološke nizove koji kodiraju protein šiljka nekih od najpoznatijih varijanti SARS-CoV-2 virusa (alfa, beta, gama i delta soj) tako da ih globalno poravnamo s nizom izvornog SARS-CoV-2 virusa. Prikazat ćemo točan broj mutacija u proteinu šiljka svakog soja u odnosu na protein šiljka izvornog soja. U ovom istraživanju mutacija se odnosi na jednu promjenu u biološkom nizu (genomu virusa). Mutacije se događaju često, ali samo ponekad mijenjaju karakteristike virusa i utječu na njegovo ponašanje. [16]

## *3.1 Alfa soj SARS-CoV-2 virusa*

Alpha soj SARS-CoV-2 virusa je varijacija originalnog SARS-CoV-2 virusa (Wuhan-Hu-1) koji je nazvan tako jer je bio prvi soj izvornog SARS-CoV-2 virusa. Alfa varijanta s deset točkastih mutacija u proteinu šiljka, bila je dva puta prenosnija od izvornog soja i širi se 56% brže u odnosu na izvorni soj. [17]

| NW Score | Identities | Positives | Gaps |
|---|---|---|---|
| 6527 | 1244/1254(99%) | 1245/1254(99%) | 3/1254(0%) |

```
Query  1    MFVFLVLLPLVSSQCVNLTTRTQLPPAYTNSFTRGVYYPDKVFRSSVLHSTQDLFLPFFS  60
Sbjct  1    ............................................................  60

Query  61   NVTWFHAI--SGTNGTKRFDNPVLPFNDGVYFASTEKSNIIRGWIFGTTLDSKTQSLLIV  118
Sbjct  61   ........HV............................................  120

Query  119  NNATNVVIKVCEFQFCNDPFLGVY-HKNNKSWMESEFRVYSSANNCTFEYVSQPFLMDLE  177
Sbjct  121  ........................Y.....................................  180

Query  178  GKQGNFKNLREFVFKNIDGYFKIYSKHTPINLVRDLPQGFSALEPLVDLPIGINITRFQT  237
Sbjct  181  ............................................................  240

Query  238  LLALHRSYLTPGDSSSGWTAGAAAYYVGYLQPRTFLLKYNENGTITDAVDCALDPLSETK  297
Sbjct  241  ............................................................  300

Query  298  CTLKSFTVEKGIYQTSNFRVQPTESIVRFPNITNLCPFGEVFNATRFASVYAWNRKRISN  357
Sbjct  301  ............................................................  360

Query  358  CVADYSVLYNSASFSTFKCYGVSPTKLNDLCFTNVYADSFVIRGDEVRQIAPGQTGKIAD  417
Sbjct  361  ............................................................  420

Query  418  YNYKLPDDFTGCVIAWNSNNLDSKVGGNYNYLYRLFRKSNLKPFERDISTEIYQAGSTPC  477
Sbjct  421  ............................................................  480

Query  478  NGVEGFNCYFPLQSYGFQPTYGVGYQPYRVVVLSFELLHAPATVCGPKKSTNLVKNKCVN  537
Sbjct  481  ..................N.........................................  540

Query  538  FNFNGLTGTGVLTESNKKFLPFQQFGRDIDDTTDAVRDPQTLEILDITPCSFGGVSVITP  597
Sbjct  541  ............................A...............................  600

Query  598  GTNTSNQVAVLYQGVNCTEVPVAIHADQLTPTWRVYSTGSNVFQTRAGCLIGAEHVNNSY  657
Sbjct  601  ............D.............................................  660
```

Slika 3.5: Prikaz globalnog poravnanja pomoću Needleman-Wunsch algoritma prvih 660 aminokiselina alpha soja i izvornog soja SARS-CoV-2 virusa.


Slika 3.6: Prikaz globalnog poravnanja pomoću Needleman-Wunsch algoritma ostatka aminokiselina alpha soja i izvornog soja SARS-CoV-2 virusa.

Na slikama 3.5 i 3.6 prikazano je globalno poravnanje pomoću online sučelja za izvođenje Needleman-Wunsch algoritam. [6]. Naziv *engl. query* označava upitni niz, u našem slučaju alpha soj SARS-CoV-2 virusa, dok se skraćeni naziv *engl. Subject* odnosi na ciljani niz, odnosno na izvorni soj SARS-CoV-2 virusa. S lijeve i desne strane od poravnanja nizova nalaze se brojevi koji označavaju položaj aminokiseline u nizu. U ovom prikazu točkice označavaju pogodak (*engl. match*), dok crvena slova označavaju razliku između upita i ciljanog niza. U slučaju globalnog poravnanja Needleman-Wunsch algoritmom, svaka praznina se broji kao mutacija za sebe. Tako se npr. u nizu alpha soja SARS-CoV-2 virusa na pozicijama $69$ i $70$ nalaze oznake aminokiselina histadin (H) i vadin (V), dok u izvornom nizu imamo dvije praznine odnosno dva brisanja, pa taj segment brojimo kao dvije mutacije alpha soja u odnosu na izvorni soj SARS-CoV-2 virusa. Također na slikama vidimo koje su još mutacije karakteristične za alpha soj SARS-CoV-2 virusa.

## 3.2 Beta soj SARS-CoV-2 virusa

Beta soj SARS-CoV-2 virusa ima $8$ točkastih mutacija u proteinu šiljka i tri brisanja u usporedbi s izvornim sojem SARS-CoV-2 virusa. Ova varijanta uključuje jedanaest mutacija u proteinu šiljka, od kojih su tri mutacije smještene na takav položaj da potencijalno pospješuju prenosivosti i infektivnost virusa, također se $1,7$ puta brže prenosi od izvornog tipa SARS-CoV-2 virusa. [21]

| NW Score | Identities | Positives | Gaps |
|---|---|---|---|
| 6544 | 1243/1254(99%) | 1244/1254(99%) | 3/1254(0%) |

```
Query  1    MFVFLVLLPLVSSQCVNFTTRTQLPPAYTNSFTRGVYYPDKVFRSSVLHSTQDLFLPFFS  60
Sbjct  1    ................L.......................................  60

Query  61   NVTWFHAIHVSGTNGTKRFANPVLPFNDGVYFASTEKSNIIRGWIFGTTLDSKTQSLLIV  120
Sbjct  61   ................D.......................................  120

Query  121  NNATNVVIKVCEFQFCNDPFLGVYYHKNNKSWMESEFRVYSSANNCTFEYVSQPFLMDLE  180
Sbjct  121  ............................................................  180

Query  181  GKQGNFKNLREFVFKNIDGYFKIYSKHTPINLVRGLPQGFSALEPLVDLPIGINITRFQT  240
Sbjct  181  ..............................D.............................  240

Query  241  L---HRSYLTPGDSSSGWTAGAAAYYVGYLQPRTFLLKYNENGTITDAVDCALDPLSETK  297
Sbjct  241  .LAL........................................................  300

Query  298  CTLKSFTVEKGIYQTSNFRVQPTESIVRFPNITNLCPFGEVFNATRFASVYAWNRKRISN  357
Sbjct  301  ............................................................  360

Query  358  CVADYSVLYNSASFSTFKCYGVSPTKLNDLCFTNVYADSFVIRGDEVRQIAPGQTGNIAD  417
Sbjct  361  ........................................................K...  420

Query  418  YNYKLPDDFTGCVIAWNSNNLDSKVGGNYNYLYRLFRKSNLKPFERDISTEIYQAGSTPC  477
Sbjct  421  ............................................................  480

Query  478  NGVKGFNCYFPLQSYGFQPTYGVGYQPYRVVVLSFELLHAPATVCGPKKSTNLVKNKCVN  537
Sbjct  481  ...E..............N...........................................  540

Query  538  FNFNGLTGTGVLTESNKKFLPFQQFGRDIADTTDAVRDPQTLEILDITPCSFGGVSVITP  597
Sbjct  541  ............................................................  600

Query  598  GTNTSNQVAVLYQGVNCTEVPVAIHADQLTPTWRVYSTGSNVFQTRAGCLIGAEHVNNSY  657
Sbjct  601  ............D...............................................  660
```

Slika 3.7: Prikaz globalnog poravnanja pomoću Needleman-Wunsch algoritma prvih 660 aminokiselina beta soja i izvornog soja SARS-CoV-2 virusa.

Isto kao i kod alpha soja mutaciju brisanja na poziciji $242 - 244$ brojimo kao ukupno tri mutacije. U nizu beta soja na tim pozicijama se nalaze aminokiseline leucin (L) i alanin (A), dok u izvornom nizu imamo tri praznine. Ako usporedimo poravnanja alpha i beta soja SARS-CoV-2 vidimo da imaju zajedničke mutacije N501Y i D614G.

```
Query  658   ECDIPIGAGICASYQTQTNSPRRARSVASQSIIAYTMSLGVENSVAYSNNSIAIPTNFTI  717
Sbjct  661   ......................................A...................  720

Query  718   SVTTEILPVSMTKTSVDCTMYICGDSTECSNLLLQYGSFCTQLNRALTGIAVEQDKNTQE  777
Sbjct  721   ..........................................................  780

Query  778   VFAQVKQIYKTPPIKDFGGFNFSQILPDPSKPSKRSFIEDLLFNKVTLADAGFIKQYGDC  837
Sbjct  781   ..........................................................  840

Query  838   LGDIAARDLICAQKFNGLTVLPPLLTDEMIAQYTSALLAGTITSGWTFGAGAALQIPFAM  897
Sbjct  841   ..........................................................  900

Query  898   QMAYRFNGIGVTQNVLYENQKLIANQFNSAIGKIQDSLSSTASALGKLQDVVNQNAQALN  957
Sbjct  901   ..........................................................  960

Query  958   TLVKQLSSNFGAISSVLNDILSRLDKVEAEVQIDRLITGRLQSLQTYVTQQLIRAAEIRA  1017
Sbjct  961   ..........................................................  1020

Query  1018  SANLAATKMSECVLGQSKRVDFCGKGYHLMSFPQSAPHGVVFLHVTYVPAQEKNFTTAPA  1077
Sbjct  1021  ..........................................................  1080

Query  1078  ICHDGKAHFPREGVFVSNGTHWFVTQRNFYEPQIITTDNTFVSGNCDVVIGIVNNTVYDP  1137
Sbjct  1081  ..........................................................  1140

Query  1138  LQPELDSFKEELDKYFKNHTSPDVDLGDISGINASVVNIQKEIDRLNEVAKNLNESLIDL  1197
Sbjct  1141  ..........................................................  1200

Query  1198  QELGKYEQYIKWPWYIWLGFIAGLIAIVMVTIMLCCMTSCCSCLKGCCSCGSCC  1251
Sbjct  1201  ......................................................  1254
```

Slika 3.8: Prikaz globalnog poravnanja pomoću Needleman-Wunsch algoritma ostatka aminokiselina beta soja i izvornog soja SARS-CoV-2 virusa.

Svih 8 mutacija i brisanja na pozicijama $242 - 244$ u beta soju SARS-CoV-2 virusa istaknuti su crvenom bojom i slovom koje označava zamjenu aminokiseline u upitnom nizu u odnosu na izvorni niz SARS-CoV-2 virusa u prikazu na slikama 3.7 i 3.8. Beta soj sadrži mutacije koje potencijalno olakšavaju pričvršćivanje na ljudske stanice zbog sljedeće tri mutacije u proteinu šiljka: N501Y, K417N i E484K.

## *3.3 Gama soj SARS-CoV-2 virusa*

Gama soj SARS-CoV-2 virusa sadrži $12$ točkastih mutacija u proteinu šiljka. Četiri je puta prenosniji od izvornog tipa SARS-CoV-2 virusa. Nagli porast broja prijema u bolnicu bio je značajan problem ove varijante. [19]

| NW Score | Identities | Positives | Gaps |
|---|---|---|---|
| 6546 | 1242/1254(99%) | 1244/1254(99%) | 0/1254(0%) |

```
Query  1     MFVFLVLLPLVSSQCVNFTNRTQLPSAYTNSFTRGVYYPDKVFRSSVLHSTQDLFLPFFS  60
Sbjct  1     ................L.T.....P.................................  60

Query  61    NVTWFHAIHVSGTNGTKRFDNPVLPFNDGVYFASTEKSNIIRGWIFGTTLDSKTQSLLIV  120
Sbjct  61    ............................................................  120

Query  121   NNATNVVIKVCEFQFCNYPFLGVYYHKNNKSWMESEFRVYSSANNCTFEYVSQPFLMDLE  180
Sbjct  121   ................D...........................................  180

Query  181   GKQGNFKNLSEFVFKNIDGYFKIYSKHTPINLVRDLPQGFSALEPLVDLPIGINITRFQT  240
Sbjct  181   .........R...................................................  240

Query  241   LLALHRSYLTPGDSSSGWTAGAAAYYVGYLQPRTFLLKYNENGTITDAVDCALDPLSETK  300
Sbjct  241   ............................................................  300

Query  301   CTLKSFTVEKGIYQTSNFRVQPTESIVRFPNITNLCPFGEVFNATRFASVYAWNRKRISN  360
Sbjct  301   ............................................................  360

Query  361   CVADYSVLYNSASFSTFKCYGVSPTKLNDLCFTNVYADSFVIRGDEVRQIAPGQTGTIAD  420
Sbjct  361   ........................................................K...  420

Query  421   YNYKLPDDFTGCVIAWNSNNLDSKVGGNYNYLYRLFRKSNLKPFERDISTEIYQAGSTPC  480
Sbjct  421   ............................................................  480

Query  481   NGVKGFNCYFPLQSYGFQPTYGVGYQPYRVVVLSFELLHAPATVCGPKKSTNLVKNKCVN  540
Sbjct  481   ...E.............N...........................................  540

Query  541   FNFNGLTGTGVLTESNKKFLPFQQFGRDIADTTDAVRDPQTLEILDITPCSFGGVSVITP  600
Sbjct  541   ............................................................  600

Query  601   GTNTSNQVAVLYQGVNCTEVPVAIHADQLTPTWRVYSTGSNVFQTRAGCLIGAEYVNNSY  660
Sbjct  601   ............D.......................................H.....  660
```

Slika 3.9: Prikaz globalnog poravnanja pomoću Needleman-Wunsch algoritma prvih 660 aminokiselina gama soja i izvornog soja SARS-CoV-2 virusa.

```
Query  661   ECDIPIGAGICASYQTQTNSPRRARSVASQSIIAYTMSLGAENSVAYSNNSIAIPTNFTI  720
Sbjct  661   ............................................................  720

Query  721   SVTTEILPVSMTKTSVDCTMYICGDSTECSNLLLQYGSFCTQLNRALTGIAVEQDKNTQE  780
Sbjct  721   ............................................................  780

Query  781   VFAQVKQIYKTPPIKDFGGFNFSQILPDPSKPSKRSFIEDLLFNKVTLADAGFIKQYGDC  840
Sbjct  781   ............................................................  840

Query  841   LGDIAARDLICAQKFNGLTVLPPLLTDEMIAQYTSALLAGTITSGWTFGAGAALQIPFAM  900
Sbjct  841   ............................................................  900

Query  901   QMAYRFNGIGVTQNVLYENQKLIANQFNSAIGKIQDSLSSTASALGKLQDVVNQNAQALN  960
Sbjct  901   ............................................................  960

Query  961   TLVKQLSSNFGAISSVLNDILSRLDKVEAEVQIDRLITGRLQSLQTYVTQQLIRAAEIRA  1020
Sbjct  961   ............................................................  1020

Query  1021  SANLAAIKMSECVLGQSKRVDFCGKGYHLMSFPQSAPHGVVFLHVTYVPAQEKNFTTAPA  1080
Sbjct  1021  ......T.....................................................  1080

Query  1081  ICHDGKAHFPREGVFVSNGTHWFVTQRNFYEPQIITTDNTFVSGNCDVVIGIVNNTVYDP  1140
Sbjct  1081  ............................................................  1140

Query  1141  LQPELDSFKEELDKYFKNHTSPDVDLGDISGINASFVNIQKEIDRLNEVAKNLNESLIDL  1200
Sbjct  1141  ................................V...........................  1200

Query  1201  QELGKYEQYIKWPWYIWLGFIAGLIAIVMVTIMLCCMTSCCSCLKGCCSCGSCC  1254
Sbjct  1201  ........................................................  1254
```

Slika 3.10: Prikaz globalnog poravnanja pomoću Needleman-Wunsch algoritma ostatka aminokiselina gama soja i izvornog soja SARS-CoV-2 virusa.

Ako izbrojimo istaknute promjene na slikama 3.9 i 3.10 vidimo da ima točno 12 mutacija u biološkom nizu gama soja SARS-CoV-2 virusa u usporedbi s izvornim sojem SARS-CoV-2 virusa. Također, možemo uočiti da alpha, beta i gama soj imaju dvije zajedničke mutacije (N501Y i D614G) koje ubrzavaju prijenos virusa između ljudi i pojačavaju njegovu infektivnost.

## *3.4 Delta soj SARS-CoV-2 virusa*

Delta soj SARS-CoV-2 virusa sa $17$ mutacija, od koji $10$ u proteinu šiljka, kao takav prvi put je prijavljen u studenom 2020. i $2,4$ puta je prenosiviji od izvornog SARS-CoV-2 virusa. Zabilježeno je da mutacija L452R (Zamjena na poziciji $452$) potencijalno povećava infektivnost delta soja SARS-CoV-2 virusa i time potiče replikaciju virusa. Mutacija D614G, koju tumačimo kao supstituciju na poziciji $614$, odnosno zamjena asparaginske kiseline u glicin, dijeli se s drugim visoko prenosivim varijantama kao što su alfa, beta i gama sojevi SARS-CoV-2 virusa, te ona značajno poboljšava otpornost virusa, što je rezultiralo većim i bržim širenjem među ljudima. [19] Poboljšana prenosivost delta varijante povezana je s kritičnim mutacijama D614G, L452R i T478K koje možemo vidjeti na slici 3.11. Zaraznost izvornog soja SARS-CoV-2 virusa otkrivenog u Wuhanu bila je $2,4-2,6$, za alpha soj zaraznost iznosi $4-5$, dok je zaraznost za delta soj $5-8$. To znači da netko zaražen delta sojem SARS-CoV-2 može zaraziti do osam drugih osoba, to nam ukazuje koliko određene mutacije potencijalno mogu utjecati na poboljšanje infektivnosti virusa. [18]

| NW Score | Identities | Positives | Gaps |
|---|---|---|---|
| 6535 | 1243/1254(99%) | 1244/1254(99%) | 2/1254(0%) |

```
Query  1     MFVFLVLLPLVSSQCVNLRTRTQLPPAYTNSFTRGVYYPDKVFRSSVLHSTQDLFLPFFS  60
Sbjct  1     ..................T.......................................  60

Query  61    NVTWFHAIHVSGTNGTKRFDNPVLPFNDGVYFASIEKSNIIRGWIFGTTLDSKTQSLLIV  120
Sbjct  61    ................................T.......................  120

Query  121   NNATNVVIKVCEFQFCNDPFLDVYYHKNNKSWMESG--VYSSANNCTFEYVSQPFLMDLE  178
Sbjct  121   ....................G.............EFR.....................  180

Query  179   GKQGNFKNLREFVFKNIDGYFKIYSKHTPINLVRDLPQGFSALEPLVDLPIGINITRFQT  238
Sbjct  181   ..........................................................  240

Query  239   LLALHRSYLTPGDSSSGWTAGAAAYYVGYLQPRTFLLKYNENGTITDAVDCALDPLSETK  298
Sbjct  241   ..........................................................  300

Query  299   CTLKSFTVEKGIYQTSNFRVQPTESIVRFPNITNLCPFGEVFNATRFASVYAWNRKRISN  358
Sbjct  301   ..........................................................  360

Query  359   CVADYSVLYNSASFSTFKCYGVSPTKLNDLCFTNVYADSFVIRGDEVRQIAPGQTGKIAD  418
Sbjct  361   ..........................................................  420

Query  419   YNYKLPDDFTGCVIAWNSNNLDSKVGGNYNYRYRLFRKSNLKPFERDISTEIYQAGSKPC  478
Sbjct  421   ............................L..........................T..  480

Query  479   NGVEGFNCYFPLQSYGFQPTNGVGYQPYRVVVLSFELLHAPATVCGPKKSTNLVKNKCVN  538
Sbjct  481   ..........................................................  540

Query  539   FNFNGLTGTGVLTESNKKFLPFQQFGRDIADTTDAVRDPQTLEILDITPCSFGGVSVITP  598
Sbjct  541   ..........................................................  600

Query  599   GTNTSNQVAVLYQGVNCTEVPVAIHADQLTPTWRVYSTGSNVFQTRAGCLIGAEHVNNSY  658
Sbjct  601   ............D.............................................  660
```

Slika 3.11: Prikaz globalnog poravnanja pomoću Needleman-Wunsch algoritma prvih 660 aminokiselina delta soja i izvornog soja SARS-CoV-2 virusa.

```
Query  659   ECDIPIGAGICASYQTQTNSRRRARSVASQSIIAYTMSLGAENSVAYSNNSIAIPTNFTI  718
Sbjct  661   ...................P......................................  720

Query  719   SVTTEILPVSMTKTSVDCTMYICGDSTECSNLLLQYGSFCTQLNRALTGIAVEQDKNTQE  778
Sbjct  721   ..........................................................  780

Query  779   VFAQVKQIYKTPPIKDFGGFNFSQILPDPSKPSKRSFIEDLLFNKVTLADAGFIKQYGDC  838
Sbjct  781   ..........................................................  840

Query  839   LGDIAARDLICAQKFNGLTVLPPLLTDEMIAQYTSALLAGTITSGWTFGAGAALQIPFAM  898
Sbjct  841   ..........................................................  900

Query  899   QMAYRFNGIGVTQNVLYENQKLIANQFNSAIGKIQDSLSSTASALGKLQNVVNQNAQALN  958
Sbjct  901   ..................................................D.......  960

Query  959   TLVKQLSSNFGAISSVLNDILSRLDKVEAEVQIDRLITGRLQSLQTYVTQQLIRAAEIRA  1018
Sbjct  961   ..........................................................  1020

Query  1019  SANLAATKMSECVLGQSKRVDFCGKGYHLMSFPQSAPHGVVFLHVTYVPAQEKNFTTAPA  1078
Sbjct  1021  ..........................................................  1080

Query  1079  ICHDGKAHFPREGVFVSNGTHWFVTQRNFYEPQIITTDNTFVSGNCDVVIGIVNNTVYDP  1138
Sbjct  1081  ..........................................................  1140

Query  1139  LQPELDSFKEELDKYFKNHTSPDVDLGDISGINASVVNIQKEIDRLNEVAKNLNESLIDL  1198
Sbjct  1141  ..........................................................  1200

Query  1199  QELGKYEQYIKWPWYIWLGFIAGLIAIVMVTIMLCCMTSCCSCLKGCCSCGSCC  1252
Sbjct  1201  ......................................................  1254
```

Slika 3.12: Prikaz globalnog poravnanja pomoću Needleman-Wunsch algoritma ostatka aminokiselina delta soja i izvornog soja SARS-CoV-2 virusa.

## *3.5 Mjera sličnosti*

Mjera sličnosti obično se izražava kao numerička vrijednost koja postaje veća kada su uzorci podataka sličniji. Često se izražava kao broj između nule i jedan gdje nula znači malu sličnost (uzorci podataka su različiti), dok vrijednost prema jedan znači veliku sličnost (uzorci podataka su vrlo slični). Postotak sličnosti nizova izračunat je pomoću Pythonovog modula Difflib i njegove ugrađene funkcije $ratio()$. Korišteni kod prikazan je u dodatku C.

| Mjera sličnosti | |
|---|---|
| alpha soj | 88,36 % |
| beta soj | 86,53 % |
| gama soj | 87,37 % |
| delta soj | 86,02 % |

Tablica 3.2: Tablica prikazuje postotak sličnosti različitih sojeva SARS-CoV-2 virusa u usporedbi s izvornim sojem SARS-CoV-2 virusa.

U tablici 3.2 vidimo da svi sojevi u usporedbi s izvornim SARS-CoV-2 sojem imaju veliki postotak sličnosti, što nas navodi na zaključak da su biološki nizovi ovih sojeva homologni. Koncept homologije odnosno zajedničkog evolucijskog podrijetla ključan je za provođenje računalne analize proteinskih i DNK nizova. Zaključujemo o homologiji kada dva biološka niza ili strukture dijele više sličnosti nego što bi se očekivalo. To nam govori da dva niza nisu nastala neovisno, nego su proizašla iz zajedničkog pretka. Zajedničko podrijetlo objašnjava vrlo veliku sličnost između nizova testiranih sojeva SARS-CoV-2 virusa u usporedbi s izvornim nizom. [24] Također u tablici 3.2 vidimo da najveći postotak sličnosti s izvornim sojem ima alpha soj SARS-CoV-2 virusa koji ima $9$ mutacija u proteinu šiljka koje su ravnomjerno raspoređene duž biološkog niza. Biološki niz delta soja SARS-CoV-2 virusa je najmanje sličan izvornom, jer akumulira više mutacija na pojedinim dijelovima proteina šiljka koje uzrokuju uočljive razlike s biološkim nizom izvornog SARS-CoV-2 virusa.

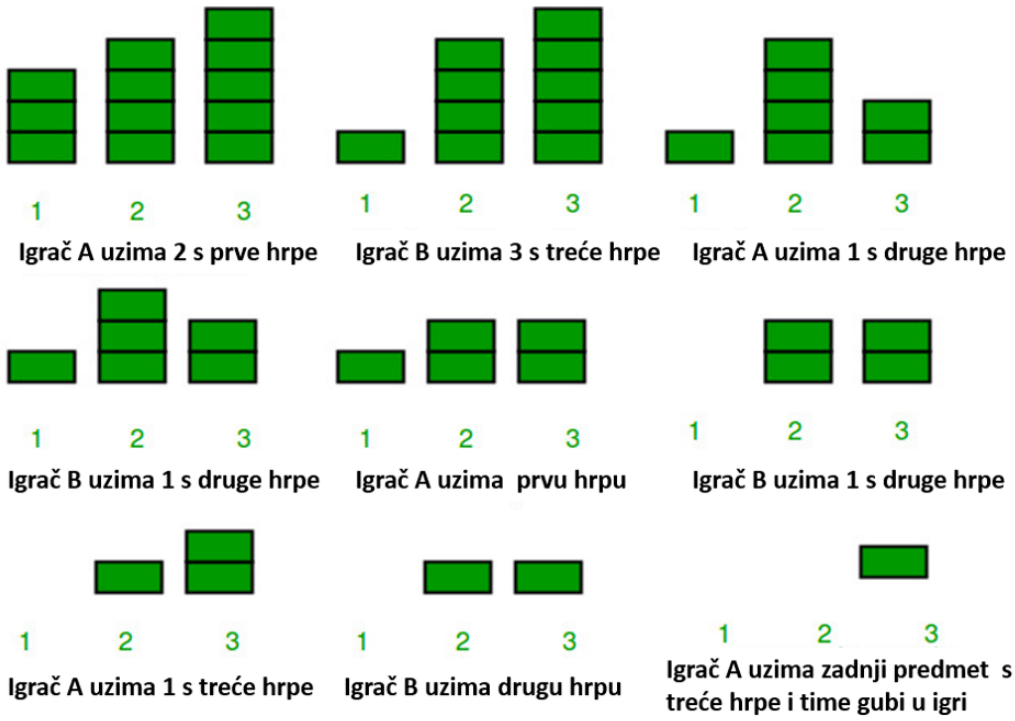
# 4 Dinamičko programiranje u srednjoj školi

Obrazovne promjene i razvoj u sustavu školstva događaju se svakodnevno. Uvođenje novih kurikuluma i obrazovnih reformi su pokrenuti kako bi se poboljšala kvaliteta obrazovanja. Prirodoslovni predmeti i predmeti iz STEM područja organizirani su tako da učenike potiču na interdisciplinarno učenje odnosno kombinaciju više područja u svrhu rješavanju nekog zadatka. Dakle, izbjegava se učenje svakog predmeta za sebe kao što je dugo bila tradicija u našem obrazovnom sustavu, već se učenike prvenstveno potiče da razmišljaju, istražuju i povezuju nova znanja i koncepte. Jedan od prijedloga je da se više obrati pozornost na matematičko i računalno razmišljanje (*engl. computational thinking*). Računalno razmišljanje se može karakterizirati kroz akritvnosti koje uključuju formuliranje i rješavanje problema na način kako bi to izvelo računalo, iako ne nužno uz pomoć računala. Smatra se neophodnom vještinom za uspješno funkcioniranje u društvu 21. stoljeća koje je prožeto tehnologijom, stoga je važna njegova integracija u obrazovni sustav. Razvoj računalnog razmišljanja nudi novu perspektivu za poboljšanje vještina učenika u pogledu matematičkih procesa modeliranja, zaključivanja i rješavanja problema. Dinamičko programiranje je važan koncept u teoriji algoritama, ali se općenito na dinamičko programiranje gleda kao na klasu algoritama koju je teško razumjeti i koja je kontraintuitivno, stoga je bolje ne objašnjavati dinamičko programiranje sa svim detaljima kao tehniku za dizajniranje algoritama nego je prikladnije rasvijetliti pojedine aspekte kako bi učenici stekli osnovno znanje i razumijevanje ovog koncepta. Također se u dinamičkom programiranju pojavljuju slične poteškoeće s kojima se nastavnik susreće kada govorimo o rekurzivnim algoritmima. Dinamičko programiranje kao paradigma sadrži mnoge aspekte računalnog razmišljanja i stoga ćemo u ovom poglavlju predložiti nekoliko aktivnosti za uvodni sat pomoću kojih možemo uvesti osnovne koncepte i pojedinačne aspekte dinamičkog programiranja u srednjoj školi.

## *4.1 Uvodna aktivnost*

Iz aktualne literature je poznato da su igre učenicima zanimljive i potiču njihovu motiviranost, stoga je cilj ove uvodne aktivnosti uključiti i pripremiti učenike za nove koncepte iz dinamičkog programiranja. Na kraju uvodnog sata o dinamičkom programiranju učenici će moći osmisliti optimalnu strategiju za pobjedu u igri pomoću di-

namičkog programiranja, objasniti što podrazumijeva dinamičko programiranje, dati primjere primjene dinamičkog programiranja te opisati osnovne koncepte. Uvodna aktivnost u ovom slučaju je strateška igra Nim (*engl. Nim-game*). Ova se igra odvija u parovima, dakle igrač A i igrač B koji naizmjenično uklanjanju $30$ predmeta s hrpe. Pravila igre je bitno pažljivo pročitati prije nego što učenici krenu igrati. Igra počinje s određenim brojem predmeta (u našem slučaju $30$), raspoređenih u jednu ili više hrpa. Prilikom svakog poteza, igrač može izabrati po jedan, dva ili tri predmeta s jedne hrpe. Igrač koji ukloni posljednji predmet s hrpe gubi u igri. Na primjer, igrač A je na redu i na stolu su tri predmeta. Ako igrač A pokupi dva predmeta sa stola, ostaje jedan na stolu. Igrač B će pokupiti preostali predmet i time gubi u igri. Jednostavna implementacija nim igre prikazana je u dodatku D.




Slika 4.13: Grafički prikaz nim igre i jedne od strategija.

Ova vježba je uvod u koncept dinamičkog programiranja. U ovoj vježbi učenici igraju igru čije su upute opisane prethodno, ali bi moglo biti korisno dati učenicima ideju o igri u objašnjenju od otprilike $2$ minute. Učenici mogu igrati ovu igru otprilike $10$ minuta. Učenici prvo igraju igru, a nakon toga smišljaju strategiju za pobjedu u opisanoj igri. Također, potrebno je napomenuti da je temeljna ideja dinamičkog programiranja rastavljanje složenih problema na manje povezane probleme. Manji problemi mogu se obično riješiti jednostavno. Nakon toga, rješenja tih manjih

problema mogu se kombinirati za rješavanje većih, složenijih problema. Vremensko trajanje ove uvodne aktivnosti je otprilike $15$ minuta.

## *4.2 Aktivnost 2*

U sljedećoj aktivnosti sugeriramo učenicima da je moguće konstruirati strategiju koja će osigurati da igrač A (početni igrač) pobijedi u igri. Dajemo im sljedeći zadatak: *Pokušajte smisliti takvu strategiju koja će osigurati pobjedu igrača A. Potrebno je zapisati niz uputa koje igrač A mora slijediti kako bi osigurao pobjedu.*
Ova se vježba fokusira na rastavljanje velikih problema na manje potprobleme i kreiranje algoritamskih rješenja. Učenicima možemo pomoći s naputkom kako mogu pronaći pobjedničku strategiju, trebali bi započeti s malim potproblemom i zatim to generalizirati na veće probleme. S tim znanjem, učenici smišljaju strategiju koja osigurava da igrač A pobijedi u igri. Drugim riječima, učenici moraju smisliti algoritamsko rješenje. Nadalje, vrlo je eksplicitno što se očekuje od učenika. Igrali su igru u uvodnoj aktivnosti i znaju što je cilj igre. Učenici zapisuju korake u istim parovima. Za rješavanje problema dinamičkog programiranja počinjemo od kraja problema. Strategija se može pronaći na sljedeći način: Igrač A pobjeđuje sa sigurnošću ako igraču B ostane jedan predmet. Dakle, igrač A mora osigurati da igraču B ostane $1$ predmet za izvlačenje, bez obzira na njegov izbor. Igrač A to može kada se na stolu nalaze $2$, $3$ ili $4$ predmeta (tada igrač A može ukloniti $1$, $2$ ili $3$ predmeta respektivno), ali s $5$ predmeta to je nemoguće. Dakle, na potezu igrača B trebalo bi biti 5 predmeta na stolu, tako da igrač A ima $2$, $3$ ili $4$ kada dođe njegov red (što se događa, kakav god bio izbor igrača B). Igrač A može osigurati da se ova situacija dogodi kada je na stolu $6$, $7$ ili $8$ predmeta, ali ne i $9$. Ovaj uzorak stalno se ponavlja, tako da dolazimo do zaključka da igrač B treba $1$, $5$, $9$, $13$, $17$, $21$, $25$ ili $29$ predmeta kako bi igrač A pobijedio. Budući da počinjemo s $30$ predmeta, igrač A se može u to uvjeriti i tako pobijediti sa sigurnošću. Učenicima treba dodatno naglasiti da kada igru započnu s 29 predmeta, igrač A ne pobjeđuje sa sigurnošću. Tada bi igrač B mogao primijeniti ovu strategiju. Koraci bi mogli izgledati ovako (pod pretpostavkom da igramo s $30$ predmeta):
1) Počinjemo s $30$ predmeta, pokupimo $1$. Zatim igrač B počinje s $29$ predmeta.
2) Pričekajte akciju igrača B. Po njegovom izboru ostat će $26$, $27$ ili $28$ predmeta. Igrač

A odabire $1$, $2$ ili $3$ predmeta tako da igrač B mora započeti svoj sljedeći potez s $25$ predmeta.
3) Ponavljamo ovo kako bi igrač B počeo s $4$ predmeta manje nego na potezu prije, tako da igrač B uvijek počinje s $1$, $5$, $9$, $13$, $17$, $21$ na stolu.
4) Budući da igrač B tada završava s $1$ predmetom na stolu, mora ga pokupiti i tako gubi u igri. Dakle, igrač A pobjeđuje.
Važno je da učenici u svojim koracima pokažu da uvijek treba biti $1, 5, 9, ..., 29$ predmeta na stolu na početku poteza igrača B. Učenici imaju deset minuta da osmisle korake slične ovima. Zatim, slijedi razredna rasprava tijekom $5$ minuta o tome koja je moguća strategija i postavljanje pitanja o pristupima kako bi provjerili jesu li učenici usvojili prvi aspekt dinamičkog programiranja. Na kraju druge aktivnosti naglasak je da učenici shvate da je važno početi od kraja, kako bi im dali ideju što se događa s $1$ predmetom i kako se situacija mijenja ako se broj predmeta poveća. Vremensko trajanje ove aktivnosti otprilike je $15$ minuta.

## *4.3 Aktivnost 3*

U aktivnosti dva smo razmatrali kako se može pronaći strategija koja osigurava pobjedu u igri. Ključno je prvo analizirati situaciju kada je jedan predmet na stolu. Pronalaženje optimalne strategije na ovaj način je primjer dinamičkog programiranja. Kod primjene koncepta dinamičkog programiranja problemi se rješavaju razlaganjem na manje potprobleme ili, na primjer, počevši ”na kraju”. Kada se rješenje dobije za manji problem, ono se zatim koristi za rješavanje većeg problema. Kako je dinamičko programiranje korišteno u aktivnosti $2$? U ovoj aktivnosti učenici moraju objasniti što je zapravo dinamičko programiranje referirajući se na aktivnost 2. Dakle, čitanje definicije dinamičkog programiranja i zatim razmišljanje o tome kako se to koristilo u prve dvije aktivnosti pomaže učenicima u razvoju vještine rastavljanja velikih problema na manje potprobleme. Nakon kratkog osvrta na aktivnosti, možemo navesti učenicima različite upotrebe dinamičkog programiranja što može utjecati na njihovu motivaciju. [23] U ovoj aktivnosti koncept dinamičkog programiranja je središnja tema. Vrijeme trajanje posljednje aktivnosti uvodnog sata je otprilike $15$ minuta.

## *4.4 Osvrt na aktivnosti*

Dinamičko programiranje je tehnika programiranja koja može biti izazovna za učenike nižih razreda srednje škole koji su novi u računalnoj znanosti ili programiranju. Koncept dinamičkog programiranja uključuje raščlanjivanje složenog problema na manje potprobleme i rješavanje svakog potproblema samo jednom, te pohranjivanje rješenja u memoriju za buduću upotrebu. Ove aktivnosti smišljene su uglavnom radi uključivanja učenika u novu tehniku te kako bi što bolje shvatili koncept dinamičkog programiranja. Postoji još mnogo igara koje su dobro poznate učenicima a u sebi kriju tehniku dinamičkog programiranja. Neke od njih su tetris, šah, blackjack, remi i mnoge druge igre. Vrlo je bitno prilikom uvođenja ovako složenih koncepata koristiti jezik koji odgovara uzrastu učenika. Poželjno je izbjegavati tehničke pojmove i žargon te koristiti jednostavan jezik za objašnjenje pojmova. Korištenjem prethodno opisanih aktivnosti možemo učenicima predstaviti dinamičko programiranje pa učenici mogu razviti izvrsne vještine i samopouzdanje.

Trenutni kurikulum nastave informatike ne uključuje detaljno izučavanje dinamičkog programiranja jer je nastava često usmjerena na osnovne metode, algoritme i modele. Tek poneki učenici završe srednju školu s nešto osnovnog znanja o dinamičkom programiranju i to najčešće učenici koji sudjeluju na natjecanjima, te se dodatno pripremaju i proučavaju različitu dodatnu literaturu, često fakultativnu. U razgovoru s nastavnicima različitih srednjih škola dolazim do saznanja da koncept dinamičkog programiranja uglavnom ne obrađuju s učenicima, ponekad s učenicima prirodoslovno-matematičkih gimnazija koji imaju fond od $105$ sati godišnje u domeni računalno razmišljanje i programiranje s ishodom *B.4.4 koristi se različitim programskim paradigmama za rješavanje problema iz stvarnoga života.* Iznimke su četverogodišnje strukovne škole zanimanja tehničar za računalstvo/računalni tehničar koji se kroz mnoštvo raznih informatičkih predmeta susretnu s konceptom dinamičkog programiranja.

# 5 Zaključak

Uspoređujući aminokiselinske nizove koji kodiraju protein šiljka u različitim inačicama SARS-CoV-2 virusa dolazimo do zaključka da različite mutacije na proteinu šiljka rezultiraju različitom kvalitetom interakcija, čime utječu na moć vezanja virusa na stanicu za svaku od varijanti. Poravnanja nizova u bioinformatici su moćan način za usporedbu srodnih bioloških nizova DNK ili proteina. Mogu se koristiti za otkrivanje i razumijevanje različitih činjenica o usklađenim biološkim nizovima, poput zajedničkog evolucijskog porijekla kao u ovom radu ili zajedničke strukturne funkcije. Bilo je pokušaja dizajniranja algoritama koji se izvode u linerarnom vremenu kako bi postali primjenjivi i za druge analize npr. analiza kromosoma i kompletnih genoma na običnim računalima. Međutim, vrijeme izvođenja dolazi na drugo mjesto kada je u pitanju točnost za otkrivanje mutacija i drugih anomalija. U ovom radu primjenili smo Needleman-Wunsch algoritma na biološke nizove koji kodiraju protein šiljka različitih sojeva SARS-CoV-2 virusa i potvrdili postojanje regija genoma koje sadrže mutacije koje utječu na svojstva virusa. Usporedbe bioloških nizova proteina šiljka različitih sojeva SARS-CoV-2 virusa pokazale su da su svi virusi međusobno slični. Iako svi aminokiselinski nizovi koji kodiraju protein šiljka pojedinog soja imaju veliki postotak sličnosti kao što se vidi u Tablici 3.2, među njima postoje velike razlike koje se odnose na prenosivost, infektivnost, otpornost na cjepivo i mnoge druge razlike koje ovise o pojedinoj mutaciji. Rezultati u Tablici 3.2 potvrđuju da je protein šiljka delta soja SARS-CoV-2 virusa najmanje sličan proteinu šiljka izvornog SARS-CoV-2 virusa po svome biološkom nizu i svim drugim značajkama, što bi moglo biti jedno od objašnjenja zašto je delta soj SARS-CoV-2 virusa uzrokovao najviše smrtnih slučajeva i mnoge druge komplikacije. Također iz Tablice 3.2 možemo uočiti da je biološki niz proteina šiljka alpha soja SARS-CoV-2 najviše sličan biološkom nizu proteina šiljka izvornog soja. Budući da je alpha soj bio prva inačica izvornog SARS-CoV-2 virusa pretpostavljamo da je imao vrlo malo promjena u kodirajućoj regiji proteina šiljka što se vidi na slikama 3.5 i 3.6 u poglavlju 3. Mutacije koje su navedene u ovom radu, u poglavlju 3 potencijalno mogu objasniti zaraznost i prenosivost SARS-CoV-2. U posljednjem poglavlju ovog rada predložili smo nekoliko aktivnosti i potencijalnih pitanja koja potiču učenike na razmišljanje tijekom uvođenja paradigme dinamičkog programiranja.

Kroz te aktivnosti stavaljamo naglasak na osnovne elemente koncepta dinamičkog programiranja kao što su razlaganje velikog problema na manje potprobleme te jedan od mogućih pristupa sklapanja rješenja-s kraja prema početku (*engl. bottom top approach*). Nakon izvršavanja predloženih aktivnosti učenici će moći osmisliti optimalnu strategiju za pobjedu u igri pomoću dinamičkog programiranja, objasniti što podrazumijeva dinamičko programiranje, dati primjere primjene dinamičkog programiranja te opisati osnovne koncepte i pojmove dinamičkog programiranja.

# Dodaci

## Dodatak A Needlman-Wunsch algoritam

```
#Funkcija za ispis matrice
def print_matrix(mat):
    for i in range(0, len(mat)):
        print("[", end = "")
        for j in range(0, len(mat[i])):
            print(mat[i][j], end = "")
            if j != len(mat[i]) - 1:
                print("\t", end = "")
        print("]\n")
gap_penalty = -2
match_award = 1
mismatch_penalty = -1
#nizovi koje uspoređujemo
seq1 = "ATGCA"
seq2 = "AGTCGT"
#Funkcija koja kreira nul-matricu za spremanje rezultata
def zeros(rows, cols):
    #definiramo praznu listu
    retval = []
    #postavljamo redove matrice
    for x in range(rows):
        #u svaki red dodajmo praznu listu
        retval.append([])
        # Postavljamo stupce matrice
        for y in range(cols):
            # dodajemo nule svakom stupcu u svakom retku
            retval[-1].append(0)
    return retval
#Funkcija koja određuje odgovarajuću vrijednost između dvije baze
```

```
def match_score(s1, s2):
    if s1 == s2:
        return match_award
    elif s1 == '-' or s2 == '-':
        return gap_penalty
    else:
        return mismatch_penalty
#Funkcija koja popunjava matricu rezultatima
def needleman_wunsch(seq1, seq2):
    # duljina dvaju sekvenci
    n = len(seq1)
    m = len(seq2)
    #Generiramo matricu u koju ćemo spremati rezultate
    score = zeros(m+1, n+1)
    # Popunjavamo prvi stupac
    for i in range(0, m + 1):
        score[i][0] = gap_penalty * i
    # Popunjavamo prvi red
    for j in range(0, n + 1):
        score[0][j] = gap_penalty * j
    # Popunjavamo ostatak matrice
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            # Jednadžba (2.4)
            match = score[i - 1][j - 1] + match_score(seq1[j-1], seq2[i-1])
            delete = score[i - 1][j] + gap_penalty
            insert = score[i][j - 1] + gap_penalty
            score[i][j] = max(match, delete, insert)
    return score

print_matrix(needleman_wunsch(seq1, seq2))
def needleman_wunsch(seq1, seq2):
    # Pohranimo duljinu dva niza
```

```
n = len(seq1)
m = len(seq2)
# Generiramo matricu nula za spremanje rezultata
score = zeros(m+1, n+1)
# Popunimo prvi stupac
for i in range(0, m + 1):
    score[i][0] = gap_penalty * i
# Popunimo prvi redak
for j in range(0, n + 1):
    score[0][j] = gap_penalty * j
# Popunjavanje ostatka matrice
for i in range(1, m + 1):
    for j in range(1, n + 1):
        # Računanje vrijednosti
        match = score[i - 1][j - 1] + match_score(seq1[j-1], seq2[i-1])
        delete = score[i - 1][j] + gap_penalty
        insert = score[i][j - 1] + gap_penalty
        # Uzimamo maksimalnu vrijednost
        score[i][j] = max(match, delete, insert)
#Kreiramo varijable u koje spremamo poravnanja
align1 = ""
align2 = ""
# Počinjemo iz doljnjeg desnog kuta
i = m
j = n
while i > 0 and j > 0: # Završavamo u gornjem desnom kutu
    score_current = score[i][j]
    score_diagonal = score[i-1][j-1]
    score_up = score[i][j-1]
    score_left = score[i-1][j]
    if score_current == score_diagonal + match_score(seq1[j-1], seq2[i-1]):
        align1 += seq1[j-1]
        align2 += seq2[i-1]
```

```
            i -= 1
            j -= 1
        elif score_current == score_up + gap_penalty:
            align1 += seq1[j-1]
            align2 += '-'
            j -= 1
        elif score_current == score_left + gap_penalty:
            align1 += '-'
            align2 += seq2[i-1]
            i -= 1
    while j > 0:
        align1 += seq1[j-1]
        align2 += '-'
        j -= 1
    while i > 0:
        align1 += '-'
        align2 += seq2[i-1]
        i -= 1
    # Moramo okrenuti nizove, jer su u obrnutom redosljedu
    align1 = align1[::-1]
    align2 = align2[::-1]
    print("Najbolja ocjena poravnanja: " , score[m][n])
    return(align1, align2)

output1, output2 = needleman_wunsch(seq1, seq2)

print(output1 + "\n" + output2)
```

```
In [22]: runfile('C:/Users/User/Desktop/diplomski/NW.py', wdir='C:/Users/User/Desktop/diplomski')
[0  -2  -4  -6  -8  -10]

[-2 1   -1  -3  -5  -7]

[-4 -1  0   0   -2  -4]

[-6 -3  0   -1  -1  -3]

[-8 -5  -2  -1  0   -2]

[-10    -7  -4  -1  -2  -1]

[-12    -9  -6  -3  -2  -3]

Najbolja ocjena poravnanja:  -3
početne sekvence:
ATGCA
AGTCGT
GLobalno poravnanje :
A-TGCA
AGTCGT
```

Slika A.1: Prikaz izlaza za Needleman - Wunsch algoritam.

# Dodatak B Smith-Waterman algoritam

```
#Ispis matrice
def print_matrix(mat):
    for i in range(0, len(mat)):
        print("[", end = "")
        for j in range(0, len(mat[i])):
            print(mat[i][j], end = "")
            if j != len(mat[i]) - 1:
                print("\t", end = "")
        print("]\n")
#Bodovanje
def match_score(s1, s2):
    if s1 == s2:
        return match_award
    elif s1 == '-' or s2 == '-':
        return gap_penalty
    else:
        return mismatch_penalty
#Funkcija koja kreira matricu nula za spremanje rezultata
def zeros(rows, cols):
```

```
    #definiramo praznu listu
    retval = []
    #postavljamo redove matrice
    for x in range(rows):
        #u svaki red dodajmo praznu listu
        retval.append([])
        # Postavljamo stupce matrice
        for y in range(cols):
            # dodajemo nule svakom stupcu u svakom retku
            retval[-1].append(0)

    return retval
#Funckija koja vraća maksimalni element matrice i njegov index
def find_max(mat,M,N):
     # početna inicijalizacija maksimalnog elementa
    maxElement = - 1
     # provjeravanje svakog elementa u matrici
    for i in range(1,N+1):
        for j in range(1,M+1):
            if (mat[i][j] > maxElement):
                maxElement = mat[i][j]
                indices=i,j
    return indices


def smith_waterman(seq1, seq2):
    # duljine početnih sekvenci
    n = len(seq1)
    m = len(seq2)
    #Generiramo matricu nula za spremanje rezulrara
    score = zeros(m+1, n+1)
    # prvi stupac postavimo na nulu
    for i in range(0, m + 1):
```

```
    score[i][0] = 0 * i
#prvi redak postavimo na nulu
for j in range(0, n + 1):
    score[0][j] = 0 * j
#popunjavamo ostatak matrice
for i in range(1, m + 1):
    for j in range(1, n + 1):
        #Računamo iznos provjeravanjem dijagonalne, gornje i lijeve ćelije
        match = score[i - 1][j - 1] + match_score(seq1[j-1], seq2[i-1])
        delete = score[i - 1][j] + gap_penalty
        insert = score[i][j - 1] + gap_penalty
        # Uzimamo maksimum od tih vrijednosti i spremamo u ćeliju
        score[i][j] = max(0,match, delete, insert)
print_matrix(score)

# Varijable u koje spremamo poravnanje
align1 = ""
align2 = ""
# Poravnanje započinjemo od maksimalnog elementa matrice bodovanja
i,j=find_max(score,m,n)
# Poravanje se izvršava sve dok je rezultat u pojedinoj ćeliji različit od nule
while score[i][j] != 0:
    score_current = score[i][j]
    score_diagonal = score[i-1][j-1]
    score_up = score[i][j-1]
    score_left = score[i-1][j]
    # računamo odakle smo stigli, tj. kreiramo put nazad
    if score_current == score_diagonal + match_score(seq1[j-1], seq2[i-1]):
        align1 += seq1[j-1]
        align2 += seq2[i-1]
        i -= 1
        j -= 1
    elif score_current == score_up + gap_penalty:
```

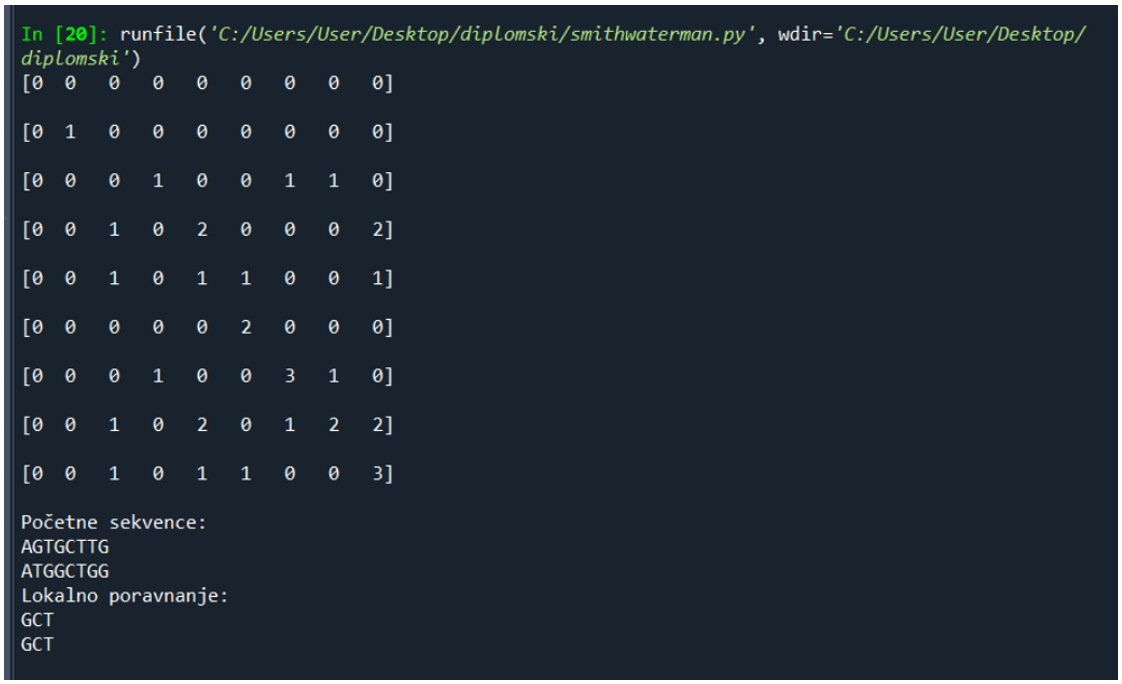
```
            align1 += seq1[j-1]
            align2 += '-'
            j -= 1
        elif score_current == score_left + gap_penalty:
            align1 += '-'
            align2 += seq2[i-1]
            i -= 1

    align1 = align1[::-1]
    align2 = align2[::-1]

    return align1,align2
#parametri
gap_penalty = -2
match_award = 1
mismatch_penalty = -1
#Nizovi koje usporedujemo
seq1="AGTGCTTG"
seq2="ATGGCTGG"

R = len(seq1)
C = len(seq2)
#Inicijalizacija matrice
matrix = []

output1, output2 = smith_waterman(seq1, seq2)
print("Početni nizovi: ")
print(seq1)
print(seq2)
print("Lokalno poravnanje: ")
print(output1 + "\n" + output2)
```




Slika B.1: Prikaz izlaza za Smith - Waterman algoritam.

# Dodatak C Mjera sličnosti

```
from difflib import SequenceMatcher
def similar(str1, str2):
    return SequenceMatcher(None, str1, str2).ratio()

f1=open("prvi_niz" , "r")
f2=open("drugi_niz" , "r")
test_string1 = f1.read()
test_string2 = f2.read()
res = similar(test_string1, test_string2) * 100
print ("Sličnost dva niza iznosi: " , res, "%")
```

# Dodatak D Nim igra

```
import random

ostalo_predmeta = random.randint(15,30)
print("Igru započinjemo s ", ostalo_predmeta ,"predmeta. \n")
uzeto_predmeta = 0
red_igraca = True
while ostalo_predmeta > 0:
    while red_igraca == True and ostalo_predmeta > 0:
        uzeto_predmeta = int(input("Koliko predmeta želiš pokupiti? "))
        if uzeto_predmeta > 3:
            print( "Možeš pokupiti maksimalno 3 predmeta!
            Trenutno stanje predmeta: " + str(stonesLeft) )
        elif ostalo_predmeta - uzeto_predmeta < 0:
            print("Ne možeš uzeti negativan broj predmeta!")
        else:
            ostalo_predmeta -= uzeto_predmeta
            print( "Pokupio si:  " + str(uzeto_predmeta) +
                " predmet(a)! Trenutno stanje predmeta: " + str(ostalo_predmeta) )
            red_igraca = False
    while red_igraca == False and ostalo_predmeta> 0:
        racunalo = random.randint( 1, min(3, ostalo_predmeta) )
        ostalo_predmeta -= racunalo
        print("\n")
        print( "Računalo uzima " + str(racunalo) +
            " predmet(a)! Trenutno stanje predmeta: " + str(ostalo_predmeta) )
        red_igraca = True

if red_igraca == True:
    print("Računalo je pokupilo zadnji predmet. Ti si pobjednik u igri!")
else:
    print("Uzeo si zanji predmet. Računalo je pobijedilo!")
```

# Bibliography


[1] Šikić, M.; Domazet-Lošo, M.: Bioinformatika, sveučilišna skripta: Zagreb, 2013.

[2] Bendelja, D.; Durgo, K.; Lukša, Ž.; Pavlica, M.; Povalec, M.: Biologija 4, Zagreb, 2021.

[3] Saenger, W.: Principles of Nucleic Acid Structure, Springer, 1984.

[4] King, J.; Kosinski-Collins, M.; Sundberg, E.: Structure and Organisation of coronaviruses // Biophysical Journal, 6 (2021), str. 1011-1033. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7682345/, 16.02.2023.

[5] Almehdi, AM.; Khoder, G.; Alchakee, AS.; Alsayyid, AT.; Sarg, NH.; Soliman, SSM. SARS-CoV-2 spike protein: pathogenesis, vaccines, and potential therapies. // Nature Public Health Emergency Collection. Vol.49, 5 (2021), str. 885-876. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8326314/, 18.02.2023.

[6] Needleman-Wunsch algoritam, National Center for Biotechnology Information. https://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE_TYPE=BlastSearch&BLAST_SPEC=GlobalAln, 23.02.2023.

[7] Blažeka, D.: Web sučelje za poravnanje sljedova. Diplomski rad. Zagreb: Fakultet elektrotehnike i računarstva, 2013.

[8] Needleman, S.; Wunsch, C.: A general method applicable to the search for similarities in the amino acid sequences of two proteins.// Journal of molecular biology. Vol. 48, 3 (1970), str. 443-453. https://pubmed.ncbi.nlm.nih.gov/5420325/, 3.02.2023.

[9] Shamir, R.: Algorithms in molecular biology, Tel Aviv University http://www.cs.tau.ac.il/~rshamir/algmb/95/94-95.html, 27.09.2022.

[10] Carroll, H.; Clement, MJ.; Ridge, P.;Snell, QO.: Effects of gap open and gap extension penalties, 2006.

[11] Grubelić, N.: Poravnanje više nizova. Diplomski rad. Zagreb : Prirodoslovno-matematički fakultet, matematički odsjek, 2015.

[12] Altschul, S. et al.: Basic Local Alignment Search Tool // Journal of Molecular Biology 215, 1990, str. 403–410. https://pubmed.ncbi.nlm.nih.gov/2231712/, 3.10.2022.

[13] Službena stranica NCBI https://blast.ncbi.nlm.nih.gov/Blast.cgi, 3.10.2022.

[14] Korber, B.; Fischer, W.; M, Gnanakaran, S.; Yoon, H.; Theiler, J.; Abfalterer, W.; Hengartner, N.; Giorgi, E.; E. Bhattacharya T.: Tracking changes in SARS-CoV-2 Spike: evidence that D614G increases infectivity of the COVID-19 virus // Cell (4) 2020, str. 812–827. https://pubmed.ncbi.nlm.nih.gov/32697968/, 10.11.2022.

[15] Sekvence varijanti spike proteina https://www.invivogen.com/, 2.1.2023.

[16] Klug, WS.; Cummings MR.; Spencer, C.; Palladino, M.: Concepts of Genetics, Global Edition 12th Edition, Pearson, 2019.

[17] Altmann, DM.; Boyton, RJ.; Beale, R.: Immunity to SARS-CoV-2 variants of concern // Science, 2021. https://www.science.org/doi/10.1126/science.abg7404, 4.01.2023.

[18] Dhawan M.; Sharma A.; Thakur N.; Rajkhowa TK.; Choudhary OP.: Delta variant (B.1.617.2) of SARS-CoV-2: Mutations, impact, challenges and possible solutions. // Hum Vaccin Immunother. Vol. 18, 5(2022). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9359381/, 28.02.2023.

[19] Mohammadi, M.; Shayestehpour, M.; Mirzaei, H.: The impact of spike mutated variants of SARS-CoV2 [Alpha, Beta, Gamma, Delta, and Lambda] on the efficacy of subunit recombinant vaccines. // The Brazilian Journal of Infectious Diseases. Vol. 25, 4 (2021). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8367756/, 7.01.2023

[20] Maaran, MR.; Bernie, RA.; Buchrieser, J.; Schwartz, O.: The Mechanism and Consequences of SARS-CoV-2 Spike-Mediated. // Journal of Molecular Bi-

ology. Vol. 434, 6 (2022), str. 2822-2836. https://pubmed.ncbi.nlm.nih.gov/34606831/, 10.01.2023.

[21] Aleem, A.; Akbar-Samad, AB.; Slenker, AK.: Emerging Variants of SARS-CoV-2 And Novel Therapeutics Against Coronavirus (COVID-19), StatPearls Publishing, 2022.

[22] Mohammad, A.; Abubaker, J.:Structural modelling of SARS-CoV-2 alpha variant (B.1.1.7) suggests enhanced furin binding and infectivity Virus Res., (2021). https://doi.org/10.1016/j.jmb.2021.167280, 20.02.2023.

[23] Slager, J.: Dynamic Programming in Dutch Secondary Education, Educational Research Project - 10 EC. // Mathematics. 5 (2021), University of Twente.

[24] Pearson WR.: An introduction to sequence similarity (”homology”) searching. // Curr Protoc Bioinformatics, 2013. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3820096/, 28.02.2023.