SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5747

Naučene indeksne strukture

Mate Paulinović

Zagreb, srpanj 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 15. ožujka 2018.

ZAVRŠNI ZADATAK br. 5747

Pristupnik: **Mate Paulinović (0036491132)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Naučene indeksne strukture**

Opis zadatka:

Indeksne strukture koristimo kada nam je potreban efikasan pristup podacima. Neke od češće korištenih su B-stabla za pristup rasponu vrijednosti, raspršeno adresiranje za pretraživanje po ključu i Bloomovi filtri za provjeru postoji li traženi zapis. Kada nam je unaprijed poznata razdioba podataka možemo tome prilagoditi odabranu strukturu i optimizirati brzinu pristupa i potrebnu memoriju, no kada želimo imati strukturu za podatke za koje unaprijed ne znamo kako će njihova distribucija izgledati, prilagodba nije moguća. U radu "The Case for Learned Indeks Structures" Tima Kraska i drugih autora, opisano je kako možemo poboljšati performanse korištenjem neuronskih mreža za učenje indeksa. U završnom radu je potrebno odabrati i implementirati jednu od opisanih naučenih struktura i napraviti usporedbu performansi s klasičnom strukturom podataka.
U radu treba koristiti neki od programskih okvira za duboko učenje kao što su Tensorflow, PyTorch i Keras.

Zadatak uručen pristupniku: 16. ožujka 2018.
Rok za predaju rada: 15. lipnja 2018.

Mentor:

Izv. prof. dr. sc. Mile Šikić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:

Prof. dr. sc. Siniša Srbljić

Zahvaljujem svojoj obitelji na konstantnoj podršci.

Zahvaljujem dr. sc. Krešimiru Križanoviću na pomoći oko tehničkih aspekata ovog rada.

Na kraju, zahvaljujem svom mentoru prof. dr. sc. Mili Šikiću na vodstvu, savjetima i pomoći.

∞

Sadržaj

Popis slika

Popis tablica

1. Uvod

Strojno učenje jedno je od najaktivnijih područja umjetne inteligencije zbog razvoja novih tehnologija koje su omogućile značajne napretke koji su donedavno bili neizvedivi. Ono se bazira na korištenju najčešće statističkih metoda za definiranje ponašanja računala bez eksplicitnog programiranja. Nalazi primjenu u brojnim disciplinama iz svih područja znanosti. Bioinformatika, znanstveno područje koje se bavi istraživanjem bioloških podataka pomoću metoda računarstva, također nije iznimka. Iznimna količina podataka koja se koristi u bioinformatici predstavlja izvrstan temelj za primjenu metoda strojnog učenja. No, osim primjena u dosad neistraženim područjima, pokazuje se kako strojno učenje može imati utjecaja i na ustanovljene algoritme i strukture podataka. Te najčešće generalne strukture se pomoću strojnog učenja mogu prilagoditi skrivenim raspodjelama podataka koje pohranjuju i time postajati efikasnije. Uz to, pokazalo se kako je rast brzine procesora usporen, a brzine grafičkih kartica su u porastu zbog razvoja novih paralelnih arhitektura, povećanja propusnosti i poboljšane skalabilnosti te su prognozirana su znatna ubrzanja u doglednoj budućnosti[1]. Na temelju toga, modeli strojnog učenja koji se baziraju na GPU operacijama bi uskoro mogli postati vrlo efikasni za izračun i samim time široko primjenjivi.
Ovaj rad bavi se jednom od fundamentalnih struktura podataka, tablicom raspršenog adresiranja. Primjenom neuronskih mreža cilj je stvoriti tablicu raspršenog adresiranja koja bolje iskorištava svoj memorijski prostor. Posebice, poboljšanje se želi ostvariti u sklopu područja bioinformatike gdje se pohranjuju sljedovi nukleotida. Problem se svodi na stvaranje strukture strojnog učenja koje bi na temelju distribucije ulaznih podataka koji se spremaju u tablicu raspršenog adresiranja naučila ravnomjernije raspodijeliti podatke po pretincima tablice, čineći ju pritom memorijski i vremenski efikasnijom.

[1] `https://www.nextbigfuture.com/2017/06/moore-law-is-dead-but-gpu-will-get-1000x-faster-by-2025.html`

2. Indeksna struktura — tablica raspršenog adresiranja

Tablica raspršenog adresiranja (engl. *hash table*) jedna je od temeljnih struktura podataka zbog operacija povoljnih vremenskih složenosti. Tablice raspršenog adresiranja koriste se u bazama podataka za indeksiranje, implementacijama spremnika privremene memorije, implementacijama skupova, prevodiocima programskih jezika itd. Ona predstavlja implementaciju *asocijativnog polja*, apstraktne strukture podataka koja će biti obrađena u nastavku.

2.1. Asocijativno polje

2.1.1. Definicija

Asocijativno polje (engl. *associative array*) definirano je u [1] i [11] kao dinamička apstraktna struktura podataka koja sprema skup parova definiranih kao

$$T = \{x := (k, e) \mid k \in K \subset U, e \in D\} \tag{2.1}$$

pri čemu je k jedinstveni ključ iz konačnog skupa korištenih ključeva K (koji je podskup skupa svih mogućih ključeva U), dok je D skup mogućih vrijednosti elementa e. Uvodi se oznaka $n = |K|$. Također, nad članovima skupa je T definiran atribut $x.ključ = k \in K$. Struktura mora omogućavati tri operacije:

- $\mathsf{POHRANA}(T, x) := T \cup \{x\}$
- $\mathsf{BRISANJE}(T, x) := T \setminus \{x\}$
- $\mathsf{PRISTUP}(T, k) := x$ akko $\exists x \in T$ takav da $x.ključ = k$

Definiramo i funkciju $h : U \to \{0, \ldots, M-1\}$ koja preslikava zadani ključ u poziciju u rasponu $[0, M-1]$ gdje je M veličina asocijativnog polja.

2.1.2. Direktno adresiranje

Definiranjem funkcije preslikavanja $h : U \to \{0, 1, \ldots, M-1\},\ h(k) = k$ i ograničavanjem skupa svih mogućih ključeva na $U = \{0, 1, \ldots, M-1\}$ te korištenjem strukture polja veličine M postižemo da se svaki par (k, d) pohranjuje jedinstveno u $T[k]$ [1]. To omogućava jednostavnu implementaciju asocijativnog polja *direktnim adresiranjem*. Direktno adresiranje prikazano je na slici 2.1. Zahtijevane operacije nad tako definiranom strukturom su trivijalne za implementaciju:

Algoritam 1 Direktno adresiranje - Pohrana

Ulaz: T – tablica direktnog adresiranja, x – par (*ključ*,*vrijednost*)

Izlaz: $\emptyset$

1: $T[h(k.ključ)] \leftarrow x$

2: **vrati**

Algoritam 2 Direktno adresiranje - Brisanje

Ulaz: T – tablica direktnog adresiranja, k – ključ

Izlaz: $\emptyset$

1: $T[h(k)] \leftarrow$ NIL

2: **vrati**

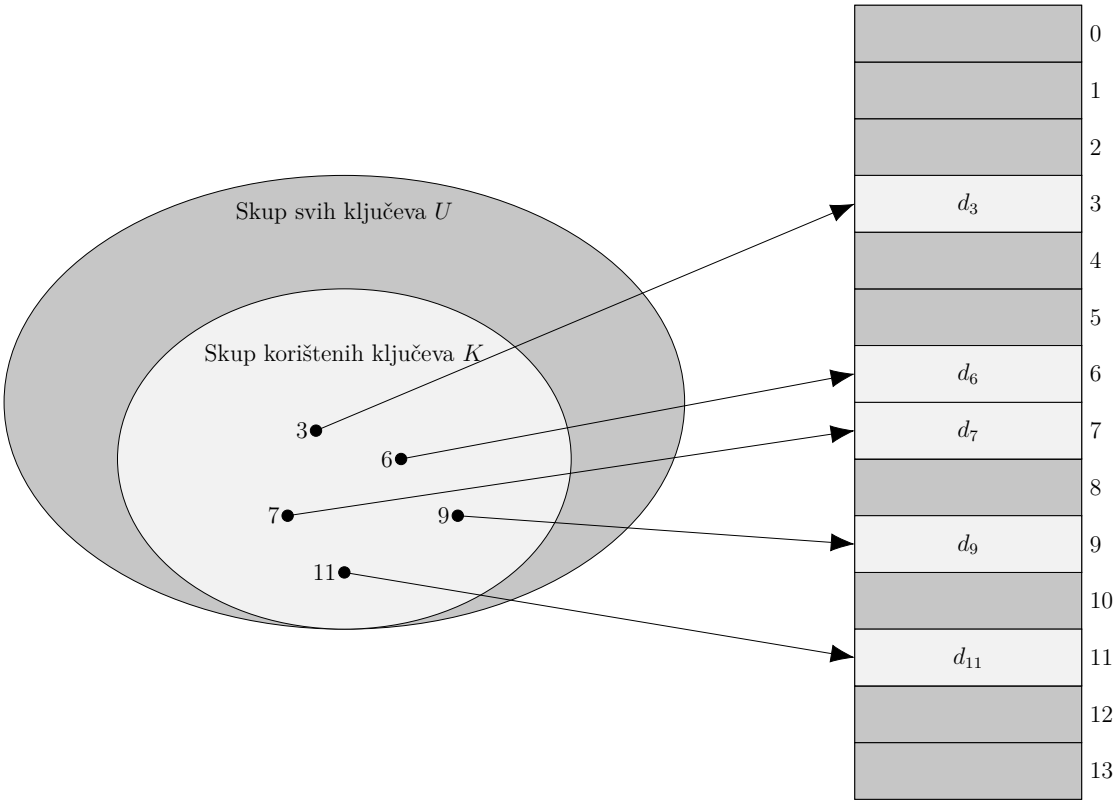
Algoritam 3 Direktno adresiranje - Pristup

Ulaz: T – tablica direktnog adresiranja, k – ključ

Izlaz: x

1: **vrati** $T[h(k)]$

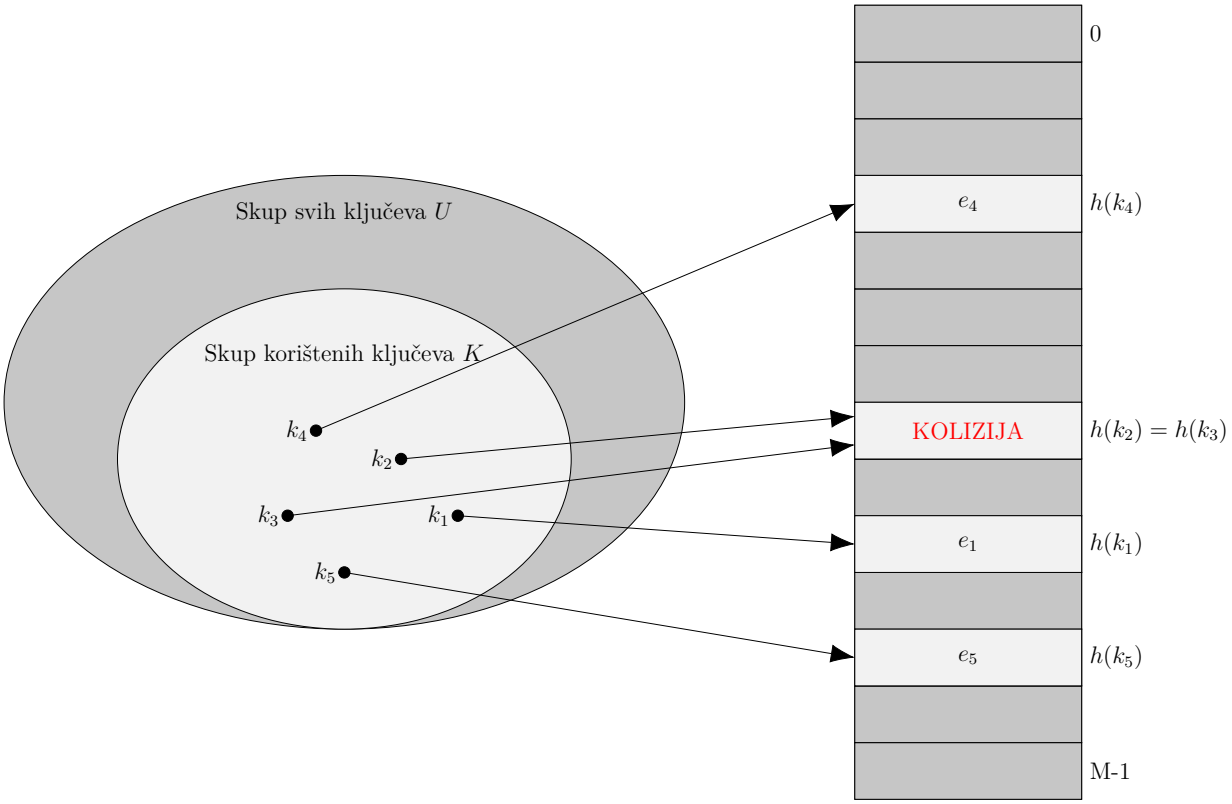
Ovako definiranje ima prednost što su sve tri operacije i za najgori slučaj složenosti $\mathcal{O}(1)$ [1]. No, kako raste broj elemenata skupa mogućih ključeva U tako mora rasti i veličina polja T. Najčešće veličina skupa mogućih ključeva je znatno veća od raspoložive veličine polja. Kao posljedica, funkcija h ne može više bijektivno preslikavati skup ključeva u indekse polja te kolizije postaju neizbježne. Također, veličina skupa K može biti znatno manja od veličine alociranog prostora polja T, zbog čega bi velika količina memorijskog prostora ostala zauzeta, a neiskorištena.



Slika 2.1: Direktno adresiranje

2.1.3. Tablica raspršenog adresiranja

Tablica raspršenog adresiranja (engl. *hash table*) može razriješiti nedostatke direktnog adresiranja navedene u 2.1.2. Za to je potrebno definirati funkciju sažimanja $h : U \to \{0, 1, \ldots, M-1\}$ koja mapira sve članove skupa svih mogućih ključeva U u pretince tablice raspršenog adresiranja. Prikaz tablice raspršenog adresiranja dan je na slici 2.2. Zbog $|U| > M$ ne može vrijediti svojstvo injektivnosti $\forall k_1, k_2 \in U \ , \ k_1 \neq k_2 \Rightarrow h(k_1) \neq h(k_2)$. Drugim riječima, postoje dva ista ključa kojima će funkcija sažimanja h pridružiti iste vrijednosti. Takav slučaj nazivamo *kolizija* te postoje brojni načini za razrješavanje kolizija koji će biti pokazani u daljnjem tekstu. Time sama efikasnost tablice raspršenog adresiranja ovisi o funkciji h i načinu rješavanja kolizija. Prednost tablice raspršenog adresiranja je garantiranjem linearne memorijske složenosti $M = \Theta(|K|)$ i dalje se zadržava konstantna vremenska složenost operacija za prosječni slučaj [11].



Slika 2.2: Tablica raspršenog adresiranja

2.2. Funkcija sažimanja

Funkcija sažimanja definira se kao funkcija koja preslikava podatke proizvoljnih dimenzija u podatke fiksnih dimenzija. Time je definiran veliki skup funkcija koje zadovoljavaju definiciju, no razlikuju se po određenim svojstvima. Funkcije sažimanja nalaze brojne primjene osim u tablicama raspršenog adresiranja, poput bloom filtara, kriptografije, traženja sličnosti, itd.

2.2.1. Svojstva funkcija sažimanja

Kako bi se zadržala poželjna svojstva tablica raspršenog adresiranja, potrebno je zadovoljiti određena svojstva „dobrih“ funkcija sažimanja s obzirom na to da je broj kolizija uvjetovan upravo načinom na koji se određuju transformacije ključeva. Različite primjene funkcija sažimanja uvjetuju različita poželjna svojstva koja bi određena funkcija trebala zadovoljiti. Posebno nas zanimaju svojstva potrebna za kvalitetno indeksiranje podataka, tj. primjenu u tablicama raspršenog adresiranja, te će takve funkcije biti razmatrane u nastavku. Valja napomenuti kako pojedina svojstva potrebna za određenu svrhu mogu biti izrazito nepovoljna

za druge primjene. Primjerice, funkcija sažimanja koju razmatramo za indeksiranje ne bi bila pogodna za primjene u kriptografiji.

Determinizam

Jedno do temeljnih svojstava je determinističnost funkcije, tj. svojstvo da će funkcija preslikati ključ k svaki put u istu vrijednost $h(k)$. Ako ova pretpostavka nije zadovoljena (preslikavanje ne ovisi samo o ključu k), gube se sve garancije o efikasnosti tablice raspršenog adresiranja, ako ne poznajemo dodatni parametar koji je korišten. Za pronalazak i brisanje elementa u tablici raspršenog adresiranja sada je potrebno pretražiti nesortirano polje, ako je funkcija sažimanja izračunala drugačiju vrijednost ključa.

Ovisnost o ključu

Prilikom izračuna vrijednosti funkcija sažimanja bi trebala koristiti sve dostupne ulazne podatke, čime se sprječava da promjene određenih dijelova ključa nemaju utjecaj na izlaznu vrijednost uzrokujući kolizije i narušavajući distribuciju izlaznih vrijednosti.

Uniformnost

Svojstvo jednostavnog uniformnog preslikavanja

$$\forall k \in U \ , \ p(h(k) = 0) = p(h(k) = 1) = \cdots = p(h(k) = M - 1) = \frac{1}{M} \qquad (2.2)$$

pri čemu je $p(h(k) = i)$ vjerojatnost da će funkcija h transformirati ključ k u vrijednost i neovisno o vrijednostima drugih ključeva [1]. Najčešće je ovu pretpostavku teško, ako ne i nemoguće provjeriti te nam je sama distribucija ključeva nepoznata. To otežava osmišljavanje funkcija sažimanja koje zadovoljavaju to svojstvo te se stoga prilikom osmišljavanja oslanja često na heurističke principe za pojedine slučajeve. Zahtijevati apsolutno uniformno preslikavanje bio bi prestrog uvjet za kvalitetu funkcije i najčešće je dovoljno da funkcija aproksimativno zadovoljava navedeno svojstvo.

Kontinuiranost

Svojstvo kontinuiranosti izlaznih vjerojatnosti zavisi ponajviše o primjeni funkcije. U slučaju indeksiranja poželjno je da funkcija slične ključeve k_1 i k_2 preslikava

u vrlo različite izlazne vrijednosti $h(k_1)$ i $h(k_2)$. Osiguravanjem tog svojstva ključevi se više raspršuju diljem tablice te se postiže ravnomjernija razdioba. Druge primjene poput traženja sličnosti pomoću funkcija sažimanja zahtijevaju da se slični ključevi preslikaju u slične izlazne vrijednosti [17].

Neinvertibilnost

Neinvertibilnost se definira kao nemogućnost određivanja ključa k na temelju izlazne vrijednosti funkcije sažimanja $h(k)$. Ovo svojstvo je bitno za kriptografske funkcije sažimanja, no za indeksiranje nije presudno.

2.2.2. Univerzalna funkcija sažimanja

Kako je već spomenuto u 2.2.1, funkcije sažimanja su osjetljive na specifičnu distribuciju ključeva te nepogodan odabir funkcije sažimanja može dovesti do brojnih kolizija i svesti složenost operacija na $\Theta(n)$ [1]. Taj se problem može razriješiti primjenom tzv. *univerzalnih funkcija sažimanja*. Ideja univerzalnih funkcija sažimanja je nasumični odabir jedne funkcije iz određene klase funkcija, što osigurava da nijedan skup ključeva neće konstantno izazivati najgori slučaj.

Definicija 2.1 ([5])**.** *Neka je c pozitivna konstanta. Klasa H funkcija sažimanja definiranih kao $h : K \to \{0, \ldots, M-1\}$ je c-univerzalna ako se bilo koja dva različita ključa k_1, k_2 preslikavaju u istu vrijednost s vjerojatnošću najviše $\frac{c}{M}$.*

$$\forall x, y \in K,\ \forall h \in H : p(h(x) = h(y)) \leq \frac{c}{M} \tag{2.3}$$

Strože svojstvo od univerzalnosti je *jaka/stroga univerzalnost*.

Definicija 2.2 ([15])**.** *Kažemo da je nasumična funkcija sažimanja $h : K \to \{0, \ldots, M-1\}$ jako/strogo c-univerzalna ako*

1. *h je c-uniforma*
   $\forall x \in K, \forall q \in \{0, \ldots, M-1\},\ p(h(x) = q) \leq \frac{c}{M}$
2. *svaki par ključeva $k_1, k_2 \in K, k_1 \neq k_2$ se izračunava nezavisno*

Također zanimljiva je poveznica univerzalnosti i uniformnosti. Iako univerzalnost ne implicira uniformnost, stroga/jaka univerzalnost to čini. Time se osigurava još jedno poželjno svojstvo te klase funkcija.

Sada možemo formalno definirati jednu klasu univerzalnih funkcija sažimanja.

Definicija 2.3 ([6]). *Neka se skup ulaznih ključeva K može predstaviti kao skup nenegativnih cijelih brojeva i neka je p prost broj veći od svih ključeva. Onda je za $a \in \{1, \ldots, p-1\}$ i $b \in \{0, \ldots, p-1\}$ klasa funkcija sažimanja $h : K \to 0, \ldots, M-1$ definiranih kao*

$$h_{a,b}(x) = ((ax + b) \bmod p) \bmod M \tag{2.4}$$

univerzalna.

2.2.3. Savršena funkcija sažimanja

Pokazano je kako je korištenjem univerzalnih funkcija sažimanja moguće smanjiti vjerojatnost postizanja najgoreg slučaja. Postavljanjem dodatnog ograničenja statičnosti na ulazni skup ključeva K moguće je osigurati složenost $\mathcal{O}(1)$ i za najgori slučaj [2]. Pošto je poznat skup ključeva koji se nalaze u tablici i nije promjenjiv moguće je osigurati injektivno preslikavanje ključeva u indekse polja, tj. nepostojanje kolizija. Pritom je korištena dvorazinska struktura tablica raspršenog adresiranja.
Prvotno se određuje vrijednost na temelju funkcije $h(k) = ((ak+b) \, mod \, p) \, mod \, M$ odabrane iz klase univerzalnih funkcija, pri čemu je p prosti broj sa svojstvom $p \geq M$. Izlazna vrijednost funkcije h određuje položaj pokazivača u tablici prve razine koji pokazuje na tablicu druge razine. Svaka tablica druge razine definira tri parametra: veličinu tablice m_j, brojeve a_j i b_j na temelju kojih se izračunava vrijednost funkcije sažimanja $h_j = ((a_j k + b_j) \, mod \, p) \, mod \, m_j$ koja je pridružena tablici raspršenog adresiranja na poziciji j. Potom se na temelju vrijednosti funkcije h_j određuje pozicija elementa u tablici raspršenog adresiranja. Pokazano je kako je moguće odrediti sve parametre funkcija sažimanja korištenih u ovakvoj strukturi u polinomijalnom vremenu [2]. Sama veličina sekundarnih tablica raspršenog adresiranja se određuje kao kvadrat broja ključeva koji su preslikani u indeks j, tj. $m_j = n_j^2$. Stvaranje ovakve strukture omogućava povoljnu vremensku složenost operacija, no povećava memorijsku složenost. Međutim, pokazano je kako se memorijska složenost cjelokupne tablice može ograničiti na $\mathcal{O}(n)$, točnije da je memorijska složenost $n + o(n)$ [1] .

2.3. Rješavanje kolizija

Spomenuto je već kako je najčešće prostor mogućih ključeva znatno veći od broja pretinaca tablice raspršenog adresiranja te se ne može ostvariti injektivno presli-

kavanje funkcije sažimanja. Iz tog razloga dolazi do kolizija, preslikavanje različitih ključeva u istu vrijednost. Kolizije narušavaju performanse tablica raspršenog adresiranja i razvijeni su različiti načini njihova razrješavanja koji će biti predstavljeni u daljnjem tekstu. Prvo definiramo neke teoreme o očekivanjima vezanim za tablicu raspršenog adresiranja.

Teorem 2.1. *Neka se sprema n različitih ključeva u tablicu raspršenog adresiranja veličine M. Pretpostavimo da je funkcija sažimanja $h : K \to \{0, \ldots, M-1\}$ 1-uniformna. Tada će očekivani broj ključeva koji se preslikavaju u jedan indeks iznositi n/M.*

Dokaz. Iz pretpostavke 1-uniformnosti funkcije h slijedi da je vjerojatnost da se ključ preslika u vrijednost i jednaka $1/M$, što može zapisati pomoću slučajne varijable $X_i = 1/M$. Očekivanje varijable X je tada suma po svim ključevima

$$X = \sum_{k=0}^{n-1} X_k = \sum_{k=0}^{n-1} \frac{1}{M} = \frac{n}{M} \tag{2.5}$$

QED

Teorem 2.2. *Neka se sprema n različitih ključeva u tablicu raspršenog adresiranja veličine M. Pretpostavimo da je funkcija sažimanja $h : K \to \{0, \ldots, M-1\}$ 1-uniformna. Tada će očekivani broj praznih pretinaca u tablici raspršenog adresiranja iznositi $M(1 - \frac{1}{M})^n$.*

Dokaz. Pridjeljivanje vrijednosti i ključu k možemo predstaviti preko Bernoullijeve sheme s obzirom na nezavisnost pohrane ključeva u tablicu raspršenog adresiranja s vjerojatnostima

$$p = \frac{1}{M},\ q = (1 - \frac{1}{M}) \tag{2.6}$$

gdje je p vjerojatnost da je neki ključ preslikan u vrijednost i, a q vjerojatnost da je preslikan u jednu od preostalih vrijednosti. Binomna slučajna varijabla X_i tada može predstavljati broj ključeva koji su preslikani u indeks i. Vjerojatnost se može izračunati preko formule

$$P(X_i = k) = \binom{n}{k} p^k q^{n-k} = \binom{n}{k} (\frac{1}{M})^k (1 - \frac{1}{M})^{n-k} \tag{2.7}$$

U slučaju kada je pretinac na poziciji i prazan ($k = 0$) uvrštavanjem u jednadžbu 2.7

$$P(X_i = 0) = \binom{n}{0} (\frac{1}{M})^0 (1 - \frac{1}{M})^n = (1 - \frac{1}{M})^n \tag{2.8}$$

Zbrajanjem vjerojatnosti za sve pretince u tablici raspršenog adresiranja dobiva se očekivanje broja praznih pretinaca

$$\sum_{i=0}^{M-1} P(X_i = 0) = \sum_{i=0}^{M-1} (1 - \frac{1}{M})^n = M(1 - \frac{1}{M})^n \tag{2.9}$$

QED

Korolar 2.2.1. *Pokazano je kako je očekivani broj praznih pretinaca u tablici raspršenog adresiranja $M(1 - \frac{1}{M})^n$. U slučaju da je broj ključeva jednak broju pretinaca, formula 2.9 prelazi u $(1 - \frac{1}{n})^n$ Kako n raste, vrijednost formule teži k e^{-1}*

$$\lim_{x \to \infty} (1 - \frac{1}{n})^n = e^{-1} \tag{2.10}$$

Iz toga slijedi, ako imamo veliku tablicu raspršenog adresiranja i vrijedi $n = M$, očekujemo n/e praznih pretinaca.

Teorem 2.3. *Neka se sprema n različitih ključeva u tablicu raspršenog adresiranja veličine M. Pretpostavimo da je funkcija sažimanja $h : K \to \{0, \ldots, M-1\}$ 1-uniformna. Tada će očekivani broj kolizija u tablici raspršenog adresiranja iznositi $n - M + M(1 - \frac{1}{M})^n$.*

Dokaz. Očekivani broj kolizija možemo intuitivno zaključiti da će biti jednak razlici broja ključeva i očekivanog broja popunjenih pretinaca tablice. Očekivani broj popunjenih pretinaca tablice može se dobiti kao razlika veličine tablice M i očekivanog broja praznih pretinaca $M(1 - \frac{1}{M})^n$

$$E(\text{kolizija}) = n - (M - M(1 - \frac{1}{M})^n) = n - M + M(1 - \frac{1}{M})^n \tag{2.11}$$

QED

Teorem 2.4 ([6])**.** *Neka se sprema n različitih ključeva u tablicu raspršenog adresiranja veličine M. Pretpostavimo da je funkcija sažimanja $h : K \to \{0, \ldots, M-1\}$ 1-uniformna. Tada je vjerojatnost da nema kolizija ograničena odozgor s $e^{-\frac{n(n-1)}{2M}}$.*

Dokaz. Kako je već prethodno pokazano, vjerojatnost da se k-ti ključ ne preslika u pretinac i je $p = 1 - \frac{k}{M}$, a vjerojatnost da nema kolizija u tablici je umnožak vjerojatnosti po svim ključevima.

$$P[\text{broj kolizija} = 0] = \prod_{k=1}^{n-1} (1 - \frac{k}{M}) \tag{2.12}$$

Koristeći nejednakost $(1-x) \leq e^{-x}$:

$$P[\text{broj kolizija} = 0] \leq \prod_{k=1}^{n-1} e^{-\frac{k}{n}} \tag{2.13}$$

$$P[\text{broj kolizija} = 0] \leq e^{-\sum_{k=1}^{n-1} \frac{k}{n}} \tag{2.14}$$

$$P[\text{broj kolizija} = 0] \leq e^{-\frac{n(n-1)}{2M}} \tag{2.15}$$

QED

Korolar 2.4.1. *Na temelju prethodnog teorema korisno je istražiti za koju vrijednost M vjerojatnost kolizije je veća od* 0.5*. Postavljamo uvjet:*

$$P[\text{broj kolizija} = 0] \leq \frac{1}{2} \tag{2.16}$$

Želimo izraziti M preko broja pohranjenih ključeva n:

$$e^{-\frac{n(n-1)}{2M}} \leq \frac{1}{2} \tag{2.17}$$

Logaritmiranjem s obje strane:

$$-\frac{n(n-1)}{2M} \leq \ln \frac{1}{2}$$

$$n^2 - n \leq 2n \ln \frac{1}{2}$$

$$n^2 - n + 2n \ln 2 \leq 0 \tag{2.18}$$

Rješenja kvadratne jednadžbe:

$$m_{1,2} = \frac{1 \pm \sqrt{8n \ln 2 + 1}}{2} \tag{2.19}$$

Zbog jednostavnosti se može aproksimirati sa:

$$m_{1,2} \approx \sqrt{2n \ln 2} \approx 1.18\sqrt{n} \tag{2.20}$$

Uvrštavanjem u 2.15:

$$P[\text{broj kolizija} = 0] \leq e^{-\frac{1.18\sqrt{n}(1.18\sqrt{n}-1)}{2n}}$$

$$P[\text{broj kolizija} = 0] \leq e^{-\frac{1.3924n-1.18\sqrt{n}}{2n}} \tag{2.21}$$

Kada bi n bio proizvoljno velik:

$$\lim_{n\to\infty} e^{-\frac{1.3924n-1.18\sqrt{n}}{2n}}$$

$$e^{\lim_{n\to\infty} -\frac{1.3924n-1.18\sqrt{n}}{2n}}$$

$$e^{-0.6962} \approx 0.498476 \tag{2.22}$$

Za tablicu raspršenog adresiranja veličine $M \approx 1.18n$, veća je vjerojatnost da će biti kolizija, nego da neće.

2.3.1. Ulančavanje

Ulančavanje je jedan od načina rješavanja kolizija. Pretinac tablice raspršenog adresiranja ne sprema sam podatak kao što je slučaj u direktnom adresiranju, već sprema pokazivač na strukturu podataka u koju se spremaju svi podatci čija se vrijednost ključa preslikala u taj pretinac. Za tu svrhu najčešće se koristi ulančana lista, no moguće su i druge strukture. Ovisno o odabranoj strukturi podataka mijenjaju se vremenske i prostorne složenosti operacija nad tablicom raspršenog adresiranja.

Ulančavanje s povezanom listom

Svakako najčešće korištena struktura podataka prilikom ulančavanje je povezana lista. U nastavku su prikazane implementacije algoritama za operacije koje tablica raspršenog adresiranja treba podržati kao implementacija asocijativnog polja.

Algoritam 4 Ulančavanje s povezanom listom - Pohrana

Ulaz: T – tablica raspršenog adresiranja, x – par (*ključ*,*vrijednost*)
Izlaz: $\emptyset$

1: $indeks \leftarrow h(x.ključ)$
2: $element.vrijednost \leftarrow x$
3: $element.sljedbenik \leftarrow T[indeks]$
4: $T[indeks] \leftarrow element$

Postoji i druga varijanta algoritma za brisanje uz korištenje dvostruko povezane liste pokazana u algoritmu 7 koja je vremenske složenosti $\mathcal{O}(1)$, no zahtijeva više memorijskog prostora.

Korisno je definirati *faktor opterećenja* α koji je potreban za analizu prostorne i vremenske složenosti.

Definicija 2.4. *Neka je n broj spremljenih ključeva u tablicu raspršenog adresiranja s ulančavanjem, a M broj pretinaca te tablice. Varijablu α definiranu kao*

$$\alpha = \frac{n}{M} \tag{2.23}$$

nazivamo faktor opterećenja.

U teoremu 2.1 pokazano je kako je očekivani broj ključeva u pretincu upravo jednak ovako definiranom faktoru opterećenja α. Ta činjenica bit će iskorištena u

Algoritam 5 Ulančavanje s povezanom listom - Brisanje

Ulaz: T – tablica raspršenog adresiranja, k – ključ

Izlaz: $\emptyset$

```
1: indeks ← h(k)
2: prethodnik ← NIL
3: trenutni ← T[indeks]
4: dok je trenutni ≠ NIL ponavljaj
5:     ako je trenutni.vrijednost.ključ = k onda
6:         prethodni.sljedbenik ← trenutni.sljedbenik
7:         oslobodi(trenutni)
8:         prekid
9:     prethodnik ← trenutni
10:    trenutni ← trenutni.sljedbenik
```

Algoritam 6 Ulančavanje s povezanom listom - Pristup

Ulaz: T – tablica raspršenog adresiranja, k – ključ

Izlaz: x ili NIL

```
1: indeks ← h(k)
2: trenutni ← T[indeks]
3: dok je trenutni ≠ NIL ponavljaj
4:     ako je trenutni.vrijednost.ključ = k onda
5:         vrati T[h(k)]
6:     trenutni ← trenutni.sljedbenik
7: vrati NIL
```

Algoritam 7 Ulančavanje s dvostruko povezanom listom - Brisanje

Ulaz: T – tablica raspršenog adresiranja, x – zapis u tablici

Izlaz: $\emptyset$

```
1: ako je x.prethodnik ≠ NIL onda
2:     x.prethodnik.sljedbenik ← x.sljedbenik
3: ako je x.sljedbenik ≠ NIL onda
4:     x.sljedbenik.prethodnik ← x.prethodnik
```

daljnjoj analizi složenosti. Performanse tablice raspršenog adresiranja s ulančavanjem ovisne su o svojstvima korištene funkcije sažimanja i složenostima operacija nad listama.

Očevidno, najgori slučaj je preslikavanje svih n ključeva u isti pretinac tablice. Ubacivanje u listu se može obaviti u $\mathcal{O}(1)$ vremenu neovisno, brisanje uz istu složenost (uz korištenje dvostruko povezane liste), dok je pristup određenom ključu sveden na operaciju pretraživanja liste koja je vremenske složenosti $\Theta(n)$.

Pogledajmo sada prosječni slučaj. Uz pretpostavku *jednostavnog uniformnog sažimanja*, tj. da je funkcija sažimanja 1-uniformna te da je konstante vremenske složenosti $\mathcal{O}(1)$. Operacije pohrane i brisanja se i dalje mogu obaviti u konstantnom vremenu [1]. Ostalo je još samo promotriti vremensku složenost pristupanja elementu u listi. Pristup se može razložiti na dva slučaja, pokušaj pristupanja elementu koji ne postoji u tablici te pristup elementu koji se nalazi u tablici. Za prvi slučaj potrebno je pretražiti k članova liste. Za prosječni slučaj i dane pretpostavke očekivani broj članova liste je pokazan u teoremu 2.1 i iznosi upravo α. U to treba uračunati još i vrijeme izračuna funkcije sažimanja koje je konstantno te se dobiva vremenska složenost $\Theta(1 + \alpha)$.

U slučaju da se pristupa elementu koji se nalazi u tablici raspršenog adresiranja prije pronalaska zadanog elementa potrebno je pretražiti k elemenata koji se nalaze prije njega (koji su zbog algoritma 4 dodani u pretinac poslije traženog elementa). Neka su nasumične indikatorske varijable dane s $X_{ij} = I\{h(x_i.ključ) = h(x_j.ključ)\}$ i označavaju da se i-ti element preslikao u isti pretinac kao i j-ti element s vjerojatnošću $1/M$ kao što je pokazano u prethodnom tekstu. Očekivani broj pretraženih elemenata je sada prosjek sume za svih n elemenata očekivanog broja elemenata koji se nalaze prije traženog ključa i taj sami ključ. Formalno zapisano:

$$E\left[\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}X_{ij}\right)\right]$$

[1] Uz pretpostavku korištenja algoritma 7. U slučaju jednostruko povezane liste složenost je jednaka složenosti pristupa.

Koristeći svojstvo linearnosti:

$$
\begin{aligned}
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}E\left[X_{ij}\right]\right)\\
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{M}\right)\\
&= 1+\frac{1}{nM}\sum_{i=1}^{n}(n-i)\\
&= 1+\frac{1}{nM}\left(\sum_{i=1}^{n}n-\sum_{i=1}^{n}i\right)
\end{aligned}
$$

Koristeći prikladni indenitet sume:

$$
\begin{aligned}
&= 1+\frac{1}{nM}\left(n^2-\frac{n(n+1)}{2}\right)\\
&= 1+\frac{n-1}{2M}\\
&= 1+\frac{\alpha}{2}-\frac{\alpha}{2n}
\end{aligned}
$$

Iz toga slijedi da je i pristup elementu koji postoji u tablici raspršenog adresiranja isto $\Theta(1+\alpha)$. Postavljanjem ograničenja da je broj pretinaca proporcionalan broju spremljenih ključeva, tj. da vrijedi $n = \mathcal{O}(M)$ što povlači $\alpha = \mathcal{O}(1)$, moguće je sve operacije obavljati u složenosti $\mathcal{O}(1)$ u prosječnom slučaju.[1] Pregled izvedenih vremenskih složenosti dan je u tablici 2.1.

Tablica 2.1: Složenosti operacija tablice raspršenog adresiranja s ulančanom listom

Operacija	Prosječni slučaj	Najgori slučaj
Pohrana	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Pristup	$\mathcal{O}(1)$	$\Theta(1+\alpha)$
Brisanje	$\mathcal{O}(1)$	$\mathcal{O}(1)$

3. Metode

3.1. K-torke

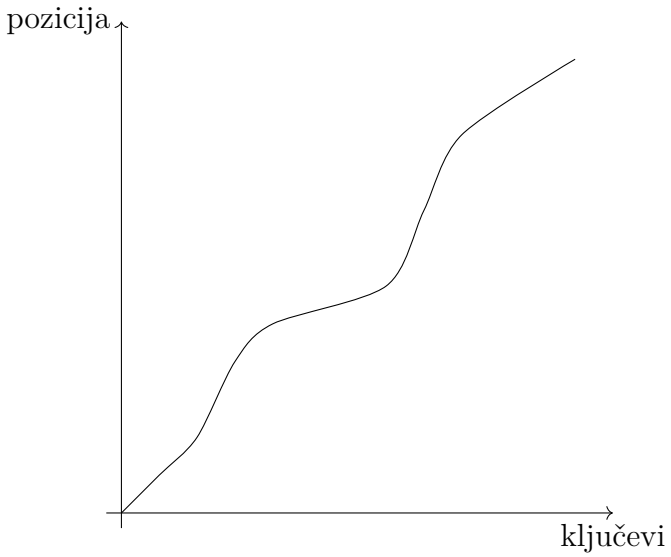
U bioinformatici najčešće se ne koriste potpuni genomi i dugački neprekinuti sljedovi nukleinskih kiselina zbog njihove dužine i nepraktičnosti. Umjesto toga koriste se $k - torke$, tj. podsljedovi nukleotida dužine k. k-torke nalaze razne primjene od metagenomike, de novo sastavljanja, otkrivanja mutacija, itd. Prikaz svih 4-torki jednog nukleinskog slijeda dan je na slici 3.1.

3.2. Funkcija sažimanja kao vjerojatnosni model

Već je ranije pokazano kako pri stvaranju funkcija sažimanja najčešće ne znamo distribuciju ključeva na koje će se primjenjivati funkcija. Različite primjene zahtijevaju različite pristupe stvaranju funkcija sažimanja, no najčešće se distribucija može zadovoljavajuće aproksimirati primjenom određenih heuristika. To ipak otvara vrata modeliranju takvih funkcija s aspekta strojnog učenja. Model strojnog učenja bi mogao naučiti distribuciju ulaznih ključeva te ju modelirati kao

ACTGTAGTCCG
ACTG
CTGT
TGTA
GTAG
TAGT
AGTC
GTCC
TCCG

Slika 3.1: Sve 4-torke jednog nukleinskog slijeda



Slika 3.2: Vjerojatnosni model ulaznog skupa ključeva

funkciju razdiobe ključeva [8]. Ograničenje postavljeno na taj ulazni skup ključeva je da posjeduju prirodni uređaj ili da se uređaj može izvesti. U najboljem slučaju takva funkcija bi se savršeno preslikala na tablicu raspršenog adresiranja i time eliminirala kolizije u tablicama. Funkcija koja aproksimira funkciju razdiobe definirana kao $F : K \rightarrow [0, 1]$ preslikava ključ u vjerojatnost $P(X <= k)$ da se pojavio ključ jednak ili manji (po korištenom uređaju ključeva). Primjenu u tablicama raspršenog adresiranja je trivijalno dobiti iz takve definicije kao:

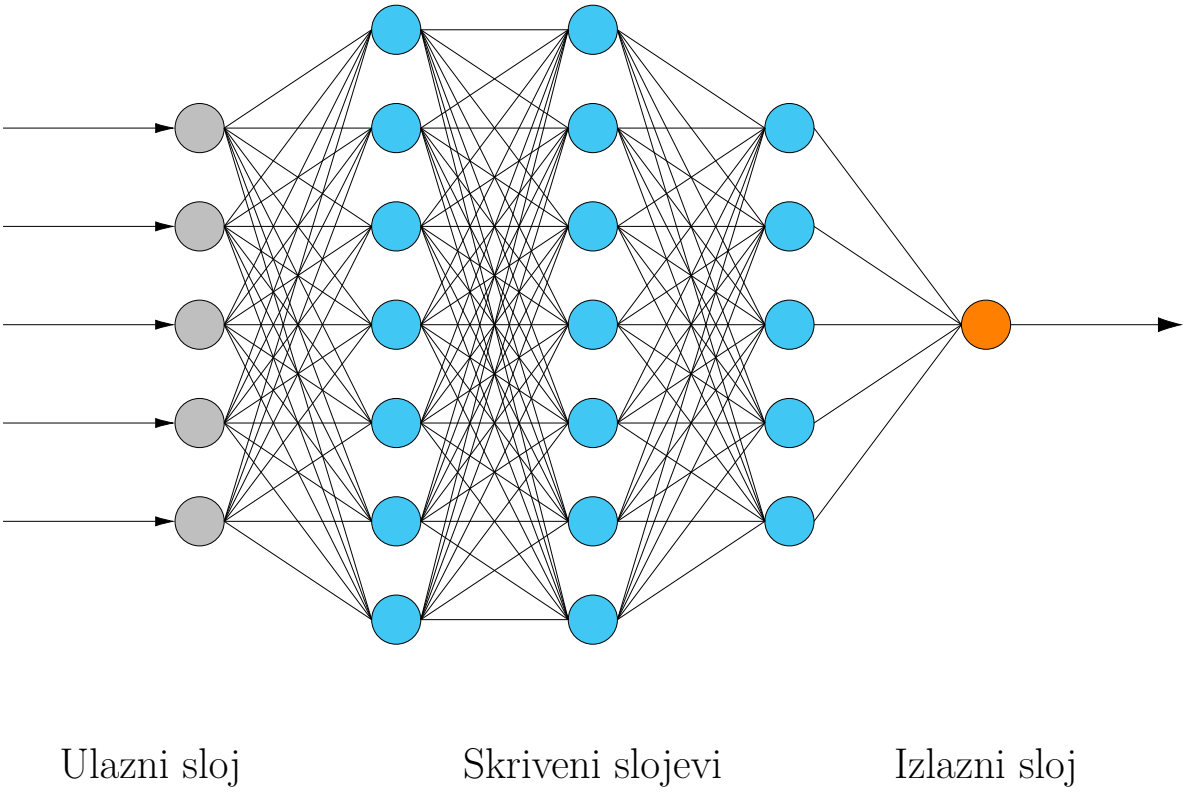
$$h(k) = F(k) \times M \tag{3.1}$$

gdje je M veličina tablice raspršenog adresiranja. Prikaz korištenja funkcije sažimanja s naučenim indeksnom dan je na slici 3.2.

3.3. Arhitektura neuronske mreže

3.3.1. Unaprijedna mreža

Unaprijedna mreža predstavlja osnovni tip neuronske mreže. Prikaz jedne takve mreže dan je na slici 3.3. Sivom bojom prikazan je ulazni sloj, plavom skriveni slojevi, a narančastom izlazni sloj. Prikazan je potpuno povezani model mreže, no neuroni unutar mreže mogu biti povezani proizvoljno. Glavna odlika ovakvog tipa mreža je da se povezanost prenosi iz jednog sloja samo u sljedeći sloj, tj. nema povratnih veza. Cilj mreže je preslikati ulazni skup X u izlazni skup Y, tj.



Slika 3.3: Unaprijedna mreža

aproksimirati stvarnu funkciju $y = f^*(x)$ parametarskim modelom $\hat{y} = f(x, \Theta)$ [7]. Cilj učenja mreže je pronaći Θ^* takav da najbolje aproksimira funkciju y. Podatci se propagiraju kroz mrežu od ulaznog sloja do izlaznog sloja slijedno, gdje svaki sloj obavlja transformaciju u matričnom računu ovisno o aktivacijskoj funkciji pojedinog neurona. U svrhu pronalaska parametra $\Theta*$ definirana je općenita funkcija cijene [7]

$$J(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\Theta}) = \frac{1}{N} \sum_{i=1}^{N} \ell(\boldsymbol{y}_i, f(\boldsymbol{x}_i, \boldsymbol{\Theta})) + \lambda \Omega(\boldsymbol{\Theta}) \tag{3.2}$$

gdje su $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}})$ funkcija gubitka i $\Omega(\boldsymbol{\Theta})$ regularizator. Na temelju takve definicije, pronalazak Θ^* se svodi na traženje parametra Θ koji minimizira funkciju J. Formalno zapisano:

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} J(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\Theta}) \tag{3.3}$$

3.3.2. Aktivacijska funkcija

Aktivacijske funkcije definiraju izlaznu vrijednost neurona na temelju ulaznih značajki, tj. hoće li za neke ulazne podatke neuron okinuti. Postoji velik broj

aktivacijskih funkcija s različitim svojstvima, no ovdje će biti predstavljene samo dvije korištene u ovom radu.

ReLU

Rampa (engl. *rectified linear unit*) je jedan on najčešće korištenih aktivacijskih funkcija koje unose nelinearnost u neuronsku mrežu, kako bi se moglo pospješiti učenje kompleksnijih značajki unutar mreže. Funkcija je definirana kao

$$ReLU(x) = \max(0, x) = \begin{cases} x, & \text{ako } x > 0, \\ 0, & \text{inače} \end{cases} \tag{3.4}$$

Izračun koristeći takvu funkciju je efikasan, no ima svojih nedostataka. Gradijent nije definiran za $x = 0$, no može se definirati da bude jednak jednostranom limesu. Također, propušta samo pozitivne ulaze, time onemogućava propagaciju gradijenta unatrag za ostale vrijednosti. Da bi se to razriješilo definirane su druge varijante koje razrješavaju navedene probleme. Njihove definicije su [3]:

$$leakyReLU(x) = \begin{cases} x, & \text{ako } x > 0, \\ kx, & \text{inače} \end{cases} \tag{3.5}$$

$$PReLU(x) = \begin{cases} x, & \text{ako } x > 0, \\ \alpha x, & \text{inače} \end{cases} \tag{3.6}$$

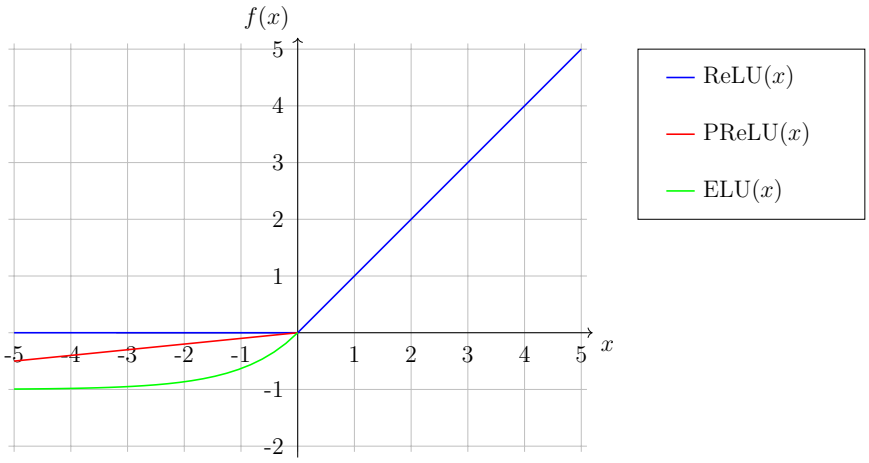
$$ELU(x) = \begin{cases} x, & \text{ako } x > 0, \\ \alpha(e^x - 1), & \text{inače} \end{cases} \tag{3.7}$$

$$SELU(x) = \lambda \begin{cases} x, & \text{ako } x > 0, \\ \alpha(e^x - 1), & \text{inače} \end{cases} \tag{3.8}$$

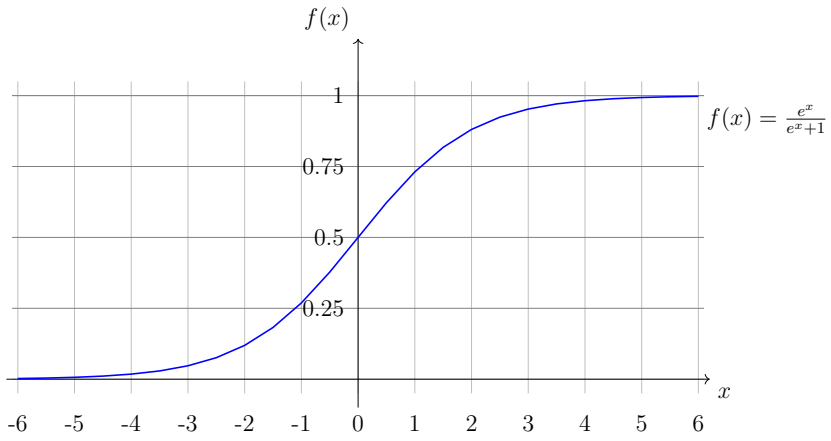
Zbog sličnosti definicija funkcija *leaky ReLU* i $PReLU$ valja naglasiti kako je kod prve konstanta $k > 0$ fiksirana, dok je kod druge funkcije parametar α učeni parametar koji se ažurira zajedno s drugim parametrima mreže. Slično vrijedi i za funkcije ELU i $SELU$, gdje kod prve α predstavlja hiperparametar, a kod druge funkcije su λ i α fiksirani parametri koji su izračunati na temelju ulaznih podataka. Grafički prikaz funkcija dan je na slici 3.4.

Sigmoida

Sigmoida je još jedna često korištena aktivacijska funkcija. Ona označava širu vrstu funkcija definiranih kao $\sigma(x) : \mathbb{R} \to [0, 1]$ ili $\sigma(x) : \mathbb{R} \to [-1, 1]$ ovisno



Slika 3.4: ReLU funkcije



Slika 3.5: Sigmoida

o korištenoj konvenciji. Često se sigmoidom naziva *logistička funkcija* koja je definirana kao [7]:

$$\sigma(x) = \frac{e^x}{e^x + 1} \tag{3.9}$$

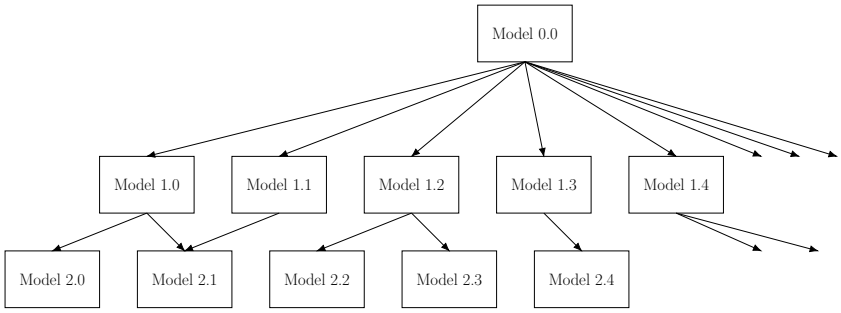
Kao i prethodno spomenuta zglobnica, sigmoida se koristi za uvođenje nelinearnosti u neuronske mreže i ograničavanje izlazne vrijednosti neurona u željeni interval. Sigmoida je prikazana na slici 3.5 i kao što je vidljivo relativno brzo se asimptotski približava vrijednostima 0 i 1 te time gradijent teži k nuli što se naziva problemom nestajućeg gradijenta (engl. *vanishing gradient*) što onemogućava učenje.

3.4. Struktura mješavine eksperata

Struktura mješavine eksperata predložena u Shazeer et al. [12] poslužila je kao temeljna ideja stvaranja modela strojnog učenja za problem učenja indeksnih struktura. Model je malo preinačen od originalno predloženog te je njegova definicija dana u nastavku. Ovakav model je nazvan *rekurzivni model indeksa* (engl. *recursive model index*) i predstavljen u Kraska et al. [8]. Slikovni prikaz zadanog modela je dan na slici 3.6. Cjelokupni model je definiran kao hijerarhijska struktura s l razina svaka s M_l modela. Model na razini i na temelju svoje izračunate vrijednosti bira sljedeći model na razini $i + 1$. Time se postiže „uža specijalizacija“ pojedinih modela u hijerarhijskoj strukturi jer je svaki model učen na temelju određenog podskupa ulaznih podataka. Time se postiže strukture usmjerenog acikličkog grafa. Svaki model ima na raspolaganju jedan ili više modela na sljedećoj razini za odabir (osim ako nije model na najnižoj razini), no svaki model nije nužno biran od samo jednog modela s više razine. Također, svaki model radi predikciju točnog položaja elementa u tablici s određenom greškom na temelju koje bira „najboljeg eksperta“ na idućoj razini koji će bolje znati odrediti položaj. Samo biranje se odvija preko funkcije:

$$k = \lfloor M_l f_{l-1}(x) \rfloor \tag{3.10}$$

gdje je l oznaka razine, M_l broj modela na razini l, f_{l-1} funkcija modela prethodnog sloja te k broj modela na razini l. Funkciju modela k na razini l označavamo s $f_l^{(k)}$. Takav zadani hijerarhijski model zahtijeva malo promijenjeni model učenja od standardnog strojnog učenja. Pregled algoritma učenja rekurzivnog modela indeksa dan je u algoritmu 8. Treniranju mreže se pristupa hijerarhijski, prvo se uče modeli najviše razine i na temelju njihovih predikcija se raspoređuju ulazne značajke koje će trenirati model na idućoj razini.



Slika 3.6: Rekurzivni model indeksa

Algoritam 8 Učenje rekurzivnog modela indeksa

Ulaz: $M[\,]$ – polje broja modela po razinama, *podatci* – ulazni skup podataka za učenje, $modeli[\,][\,]$ – polje netreniranih modela

Izlaz: $modeli[\,][\,]$ – polje treniranih modela

1: $l \leftarrow M.veličina$
2: $priv\_podaci[\,][\,]$
3: $priv\_podaci[0][0] \leftarrow podatci$
4: **za svaki** $i \in \{0, \ldots, l-1\}$ **ponavljaj**
5: **za svaki** $j \in \{0, \ldots, M[i]\}$ **ponavljaj**
6: $modeli[i][j] \leftarrow$ nova neuronska mreža trenirana na $priv\_podaci[i][j]$
7: **ako je** $i < l$ **onda**
8: **za svaki** $r \in priv\_podaci[i][j]$ **ponavljaj**
9: $p \leftarrow \lfloor f_i^{(j)}(r.ključ) M[i+1] \rfloor$
10: $priv\_podaci[i+1][p] \leftarrow r$
11: **vrati** *modeli*

3.5. Stohastičko učenje funkcije razdiobe

Stohastičko učenje funkcije razdiobe metoda je predložena u radu Magdon-Ismail i Atiya [10]. Metoda je pogodna samo za aproksimaciju univarijatnih funkcija razdiobe, što je za potrebe ovog rada prihvatljivo. Prvo potrebno je dokazati sljedeći teorem koji je korišten u definiciji same metode.

Teorem 3.1 ([14])**.** *Neka je U uniformna slučajna varijabla u intervalu $[0,1]$ i neka je X slučajna varijabla sa strogo rastućom funkcijom razdiobe $F_X(x)$. Tada slučajna varijabla $Y = F_X^{-1}(U)$ ima istu razdiobu kao i X, dok je slučajna varijabla $Z = F_X(X)$ uniformno distribuirana po intervalu $[0,1]$.*

Dokaz.

Počevši od razdiobe varijable Y:

$$\begin{aligned} F_Y(x) &= p(Y \leq x) \\ &= p(F_X^{-1}(U) \leq x) \\ &= p(U \leq F_X(x)) \\ &= F_X(x) \end{aligned} \tag{3.11}$$

pošto je $\forall x,\ F_X(x) \in [0,1]$ i $\forall y \in [0,1],\ p(U \leq y) = y$. Iz toga slijedi da X i Y imaju istu razdiobu.

Uzmimo sad slučajnu varijablu Z:

$$\begin{aligned} F_Z(x) &= p(Z \leq x) \\ &= p(F_X(X) \leq x) \\ &= p(X \leq F_X^{-1}(x)) \\ &= F_X(F_X^{-1}(x)) \\ &= x \end{aligned} \tag{3.12}$$

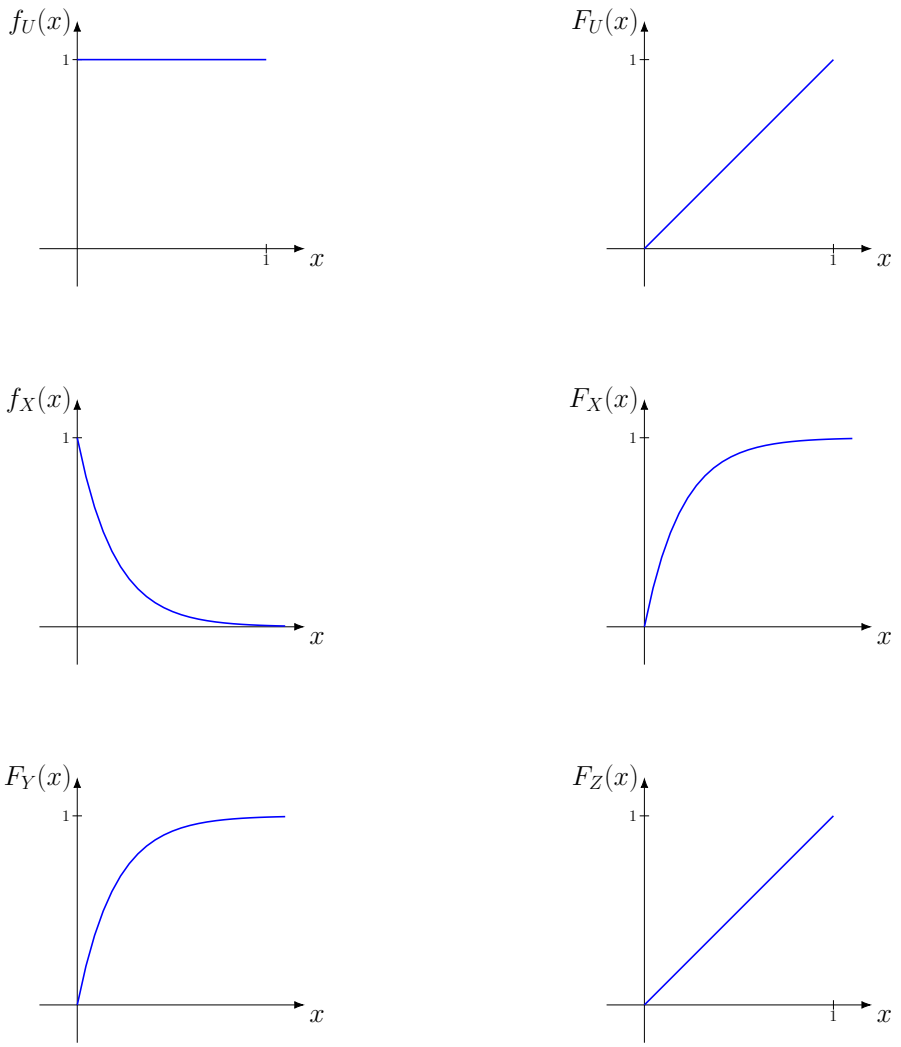
što vrijedi $\forall x \in [0,1]$ na temelju čega je Z uniformna slučajna varijabla na [0,1].
QED

Definicija 3.1. *Neka su $x_n \in \mathbb{R}, n = 0, \ldots, N-1$ pojedinačni ulazni podaci. Neka je funkcija gustoće vjerojatnosti $g(x)$ koju želimo odrediti te funkcija razdiobe $G(x) = \int_{-\infty}^{x} g(x)$. Neka je izlaz neuronske mreže $H(x, \Theta)$ gdje su x ulazni podatci, a Θ težine mreže. Po teoremu 3.1 vrijedi da je distribucija slučajne varijable $Z = G(x)$ uniformna u intervalu $[0,1]$. Ako izlazne vrijednosti neuronske mreže imaju uniformnu distribuciju, onda bi funkcija $H(x, \Theta)$ trebala dobro predstavljati funkciju $G(x)$.*

Nadalje definirana je sljedeća funkcija cijene

$$J = \sum_{n=1}^{N} \left[H(x_n) - v_n\right]^2 + \lambda \sum_{k=1}^{N_h} u\left(H(y_k) - H(y_k + \Delta)\right) \left[H(y_k) - H(y_k + \Delta)\right]^2 \tag{3.13}$$

korištena u procesu učenja. Prvi član je srednja kvadratna pogreška, a drugi član regularizator koji se uvodi za osiguranje monotonosti [13]. Točke za koje želimo osigurati monotonost su označene s y_k, dok je funkcija u Heavisideova funkcija. Sam postupak učenja je dan algoritmom 9. Prvo se ulazni skup podataka sortira uzlazno tako da je zadovoljeno $x_0 \leq x_1 \leq \ldots \leq x_{N-1}$ (linija 1). Potom se generira



Slika 3.7: Prikaz funkcija gustoće i razdiobe slučajnih varijabli iz teorema 3.1

skup od N brojeva iz uniformne razdiobe na intervalu $[0, 1]$ koji se potom sortiraju tako da vrijedi $y_0 \leq y_1 \leq \ldots \leq y_{N-1}$ (linije 3 i 4). Potom se provodi jedna iteracija učenja koristeći prethodno definiranu funkciju cijene. Učenje se provodi t_{max} puta te se za ulazni skup X svaki put generira različiti skup labela V (jer se vjerojatnosti nasumično generiraju iz uniforme distribucije) čime se postiže efekt zaglađivanja. Prilikom svake iteracije učenja težine neuronske mreže se ažuriraju na temelju algoritma propagacije unatrag (linija 5), gdje se težina mijenja ovisno derivaciji funkcije cijene o pojedinim težinama pomnožene s faktorom učenja η. Sami detalji implementacije, kao što su veličina faktora učenja, korištenje momenta, faktor propadanja ovise o korištenom algoritmu optimizacije.

Algoritam 9 Stohastičko učenje funkcije razdiobe

Ulaz: $X[\,]$ – polje ulaznih podataka, Θ_0 – početne vrijednosti težina neuronske mreže, t_{max} – broj iteracija učenja

Izlaz: Θ – krajnje vrijednosti težina neuronske mreže

1: $sortiraj(X)$
2: **za svaki** $t \in \{0, \ldots, t_{max} - 1\}$ **ponavljaj**
3: $\quad Y \leftarrow generirajUniformnuRazdiobu(0, 1, X.veličina)$
4: $\quad sortiraj(Y)$
5: $\quad \Theta_{t+1} \leftarrow \Theta_t - \eta_t \frac{\partial J}{\partial \Theta}$
6: **vrati** $\Theta_{t_{max}-1}$

4. Implementacija

4.1. Ideja rješenja

Implementacija nastoji poboljšati memorijsko korištenje tablice raspršenog adresiranja. Temeljna ideja implementacije može se razložiti na tri dijela:

1. Obrada podataka
2. Učenje modela
3. Korištenje modela

Prvi korak je obrada podataka koji se sastoji od uzimanja zapisa podataka i njihova prevođenja u oblik koji će kasnije biti korišten za učenje. Za ovaj rad to su zapisi formata FASTA i FASTQ. Svi zapisi se očitaju te se iz njih izvuku sve k-torke koje će činiti značajke našeg kasnijeg učenja. Sve k-torke se ne predstavljaju više kao nizovi slova, već se zapisuju pomoću brojeva (formata koji je potreban za učenje) koristeći preslikavanje koje je definirano u daljnjem tekstu. Potom slijedi definiranje prethodno spomenutog rekurzivnog modela indeksa koji se trenira metodom stohastičkog učenja funkcije razdiobe. Trenirani model se potom serijalizira te koristi.

4.2. Tehnologije

Sav kod napisan je u programskom jeziku Python[1] verzije 3.6.1. Python je odabran prvenstveno zbog podrške biblioteka za strojno učenje koje su bile korištene u radu. Osim Pythona korišten je i PyTorch[2] okvir za strojno učenje verzije 0.1.12 te BioPython[3] paket za bioinformatiku verzije 1.71. Točne upute za preuzimanje

[1] `https://www.python.org`
[2] `https://pytorch.org/`
[3] `https://biopython.org/`

koda i potrebnih ovisnosti te pokretanje se nalaze na Github repozitoriju [4] pod MIT [5] licencom.

4.3. Formati podataka

Podatci korišteni u radu nosili su informacije o zadanim nukleotidnim sljedovima. U području bioinformatike najčešće korištena dva formata su *fasta* i *fastq* koji će biti opisani u nastavku.

FASTA

FASTA format služi za pohranu podataka o sljedovima nukleotida i aminokiselina. Propisani format je vrlo jednostavan, svakom slijedu prethodi opisna linija koja započinje znakom „> “. Pošto nas samo zanimaju sljedovi nukleotida, popis važećih znakova u FASTA formatu dan je u tablici 4.1 [16].

FASTQ

FASTQ format drugi je format za pohranu sljedova nukleotida. Sličan je prethodno spomenutom FASTA formatu, no uz same nukleotide pohranjuju i vrijednosti kvalitete očitanja za svaku bazu. Format se sastoji od prve linije koja započinje znakom „@ “nakon kojeg slijedi identifikator slijeda. Sljedeća linija je sam slijed nukleotida, nakon kojeg je linija koja započinje znakom „+ “nakon koje ide izborno opet identifikator. Zadnja linija sadrži kvalitetu očitanja predočenu ASCII znakovima od 33 do 126, s tim da veći broj označava veću kvalitetu.

[4] `https://github.com/MatePaulinovic/Learned_index_structures`
[5] `https://en.wikipedia.org/wiki/MIT_License`

Tablica 4.1: Podržani kodovi nukleotida FASTA formata

Kod	Značenje
A	adenin
C	citozin
G	gvanin
T	timin
U	uracil
R	A ili G (purini)
Y	C ili T ili U (pirimidini)
K	G ili T ili U (ketoni)
M	A ili C
S	C ili G
W	A ili T ili U
B	C ili G ili T ili U
D	A ili G ili T ili U
H	A ili C ili T ili U
V	A ili C ili U
N	A ili C ili G ili T ili U
-	praznina nepoznate dužine

4.4. Funkcija sažimanja i tablica raspršenog adresiranja

Funkcija sažimanja korištena u radu definirana je u Li [9] odakle je preuzeta i definicija. Ova funkcija sažimanja spada u funkcije temeljenje na heuristici i služi samo za transformaciju ključeva koji su k-torke nukleotidnih sljedova.

Definicija 4.1. *Neka je* $\Sigma = \{\text{A,C,G,T}\}$ *abeceda nukleotida. Niz nukleotida je definiran kao* $s = a_1 a_2 \ldots a_n$ *pri čemu je* $a_i \in \Sigma$. *Funkcija preslikavanja nukleotida dana je kao:*

$$\phi'(\text{A}) = 0$$
$$\phi'(\text{C}) = 1$$
$$\phi'(\text{T}) = 2$$
$$\phi'(\text{G}) = 3$$

Funkcija sažimanja niza nukleotida je tada:

$$\phi(s) = \phi'(a_1) \times 4^{n-1} + \phi'(a_2) \times 4^{n-2} + \ldots + \phi'(a_n) = \sum_{i=1}^{n} \phi'(a_i) \times 4^{n-i} \quad (4.1)$$

Tablica raspršenog adresiranja implementirana je s ulančavanjem za rješavanje kolizija. Implementacija tablice je ortogonalna na samu izvedbu funkcije sažimanja koja se koristi za određivanje indeksa koji pripada određenom ključu. Također tablica tijekom rada vodi zapis o broju popunjenih pretinaca, broju praznih pretinaca te broju kolizija koje su se dogodile.

4.5. Model strojnog učenja

Implementiran je rekurzivni model indeksa s dvije razine. Na prvoj razini se nalazi samo jedna neuronska mreža s dva skrivena sloja, a na drugoj razini 10 000 neuronskih mreža bez skrivenih slojeva. Duboka mreža na prvoj razini je postavljena kako bi mogla bolje naučiti distribuciju ključeva te donijeti bolju odluku kojem ekspertu predati određivanje točne pozicije. Širina neuronskih mreža na obje razine je podesiva za ulazni i skrivene slojeve, no fiksirana za izlazni (gdje je samo jedan neuron). Širina ulaznog sloja ovisi o dužini k-torki s kojima se želi raditi, što je za testni primjer $k = 16$. Širina skrivenog sloja mreže prvog sloja je u testnom primjeru 32. Također slojevi su aktivirani preko ReLU funkcija kako bi se uvela nelinearnost u sami model, dok izlazni sloj ostvaruje sigmoidnu aktivacijsku funkciju kako bi izlaz mreže bio ograničen u intervalu [0,1]. S obzirom na to da se radi s ljudskim genomom u testnom primjeru, zbog njegove veličine vršnu mrežu trenira se u jednoj epohi kako ne bi došlo do pretreniranosti mreže. Kako gornja mreža dijeli skup ulaznih značajki, modeli na drugoj razini se treniraju na većem broju epoha kako bi imali dovoljno podataka za učenje. Algoritam korišten za optimizaciju je *RMSprop*[4], algoritam gradijentnog spusta s prilagođavajućom stopom učenja η. RMSprop korišten je s vrijednostima stope učenja $\eta = 0.01$, momenta 0.1 i faktora zaglađivanja 0.9.

5. Rezultati

Svi rezultati dobiveni su na računalu s Intelovim i7-4770K @ 3.50GHz procesoru s Linux Gentoo 4.1.12 operacijskim sustavom. Korištena su dva potpuna ljudska genoma:

- Genome Reference Consortium Human Build 37 (GRCh37) ~ 3.14 Gb[1]
- Genome Reference Consortium Human Build 38 (GRCh38) ~ 3.26 Gb

Podatci su dobiveni korištenjem Python paketa *Pympler*[2] i *memory_profiler*[3] za memorijske podatke te ugrađeni *cProfile* za vremenske podatke.

5.1. Očekivani rezultati

Prvo na temelju teorema izvedenih u odjeljku 2.3 možemo izvesti očekivane vrijednosti kolizija, broja zauzetih i slobodnih mjesta. Pregled tih vrijednosti dan je u tablici 5.1.

5.2. Eksperimentalni rezultati

Kao što je već spomenuto korištena je RMI struktura s jednom mrežom u prvom sloju i 10 000 mreža u drugom sloju. RMI struktura trenirana je na genomu

[1] 1Gb = 1,000,000 parova baza

[2] `https://pythonhosted.org/Pympler/`

[3] `https://pypi.org/project/memory_profiler`

Tablica 5.1: Očekivane vrijednosti u tablici raspršenog adresiranja za 10M ključeva

Veličina tablice	Broj kolizija	Broj praznih pretinaca	Broj punih pretinaca
$7.5M$	4 476 978	1 976 978	5 523 022
$10M$	3 678 794	3 678 794	6 321 206
$12.5M$	3 116 612	5 616 611	6 883 389

GRCh37. Potom su pohranjivani zapisi iz genoma GRCh37 i GRCh38 u tablicu raspršenog adresiranja veličine 10M. Rezultati pohranjivanja 10M ključeva su prikazani u tablici 5.2. Ključevi su pohranjivani u tablice raspršenog adresiranja veličine 75%, 100% i 125% broja ključeva. U zagradama su dana odstupanja od očekivanih vrijednosti iz tablice 5.1. Vidljivo je kako je učeni model bliže očekivanim vrijednostima te ima povoljnije rezultate. U tablici 5.3 dan je prikaz razlika između funkcije sažimanja i učenog modela na istom testnom skupu k-torki. Vidljivo je poboljšanje u smanjenju broja kolizija neovisno o veličini tablice raspršenog adresiranja. Prema očekivanjima najveće postotno smanjenje broja kolizija je prisutno kod tablice jednake veličine skupu ključeva.

Tablica 5.2: Stvarne vrijednosti u tablici raspršenog adresiranja za 10M ključeva

Podtablica A: Funkcija sažimanja

Veličina tablice	Broj kolizija	Broj praznih pretinaca	Broj punih pretinaca
$7.5M$	4 693 049 (4.83%)	2 193 049 (10.93%)	5 306 951 (−3.92%)
$10M$	4 057 345 (10.29%)	4 057 345 (10.29%)	5 942 655 (−5.99%)
$12.5M$	3 319 900 (6.52%)	5 819 900 (3.62%)	6 680 100 (2.95%)

Podtablica B: Model

Veličina tablice	Broj kolizija	Broj praznih pretinaca	Broj punih pretinaca
$7.5M$	4 486 570 (0.21%)	1 986 570 (0.49%)	5 513 430 (−0.17%)
$10M$	3 700 791 (0.60%)	3 700 791 (0.60%)	6 299 209 (−0.35%)
$12.5M$	3 149 184 (1.05%)	5 649 184 (0.58%)	6 850 816 (0.47%)

Tablica 5.3: Razlika funkcije sažimanja i modela

Veličina tablice	Broj kolizija	Broj praznih pretinaca	Broj punih pretinaca
$7.5M$	206 479 (4.40%)	206 479 (9.42%)	−206 479 (−3.89%)
$10M$	356 554 (8.79%)	356 554 (8.79%)	−356 554 (−6.00%)
$12.5M$	170 716 (5.14%)	170 716 (2.93%)	−170 716 (−2.56%)

Vremenske karakteristike pohrane 10M zapisa u tablicu raspršenog adresiranja prikazane su u tablicama 5.4 i 5.5. Vrijeme pohrane koristeći model je značajno duže od vremena funkcije sažimanja. Kao što je prikazano u tablici 5.4 vrijeme pohrane preko PyTorch modela je ∼18 puta sporije, no valja primijetiti broj funkcijskih poziva napravljenih prilikom pohrane. Zbog korištenja biblioteke

PyTorch dolazi do značajnog dodatka (engl. *overhead*) koji rezultira s ∼9 puta više funkcijskih poziva što dodatno usporava pohranu u tablicu. Model prebačen iz biblioteke *PyTorch* u biblioteku *Numpy* ima znatno bolje vrijeme izvođenja koje je samo ∼3.6 puta sporije što je značajano poboljšanje. Razlika u brzinama vidljiva je i u tablici 5.5 gdje su prikazane srednje vrijednosti poziva funkcije pohrane i poziva funkcije izračuna pozicije zapisa. Funkcija sažimanja opisana u 4.4 svodi se na k operacija potenciranja, množenja i zbrajanja, što je svakako jednostavnije od izračuna koristeći strukture neuronskih mreža koje se provode u matričnom računu. Dodatno vremensko poboljšanje moguće je postići pojednostavljivanjem mreže kao što je smanjenje broja neurona ili broja skrivenih slojeva (u mreži prve razine), no to može dovesti do povećanja broja kolizija ako model nije u stanju dovoljno dobro naučiti distribuciju podataka.

Tablica 5.4: Vremenske karakteristike pohrane u tablicu raspršenog adresiranja

Podtablica A: Ukupno trajanje pohrane 10M zapisa

Funkcija sažimanja	Model (PyTorch)	Model (Numpy)
119.770s	2176.297s	432.129s

Podtablica B: Broj funkcijskih poziva

Funkcija sažimanja (μs)	Model (PyTorch) (μs)	Model (Numpy) (μs)
224 189 882	2 115 296 116	319 152 858

Tablica 5.5: Srednje vremenske vrijednosti pohrane u tablici raspršenog adresiranja

Operacija	Funkcija sažimanja (μs)	Model (PyTorch) (μs)	Model (Numpy) (μs)
Pohrana jednog zapisa	10.8844	214.6492	40.8861
Izračun pozicije u tablici	8.9909	210.4671	37.9976

6. Zaključak

Cilj rada bio je razviti strukturu strojnog učenja koja bi poboljšala memorijsku iskorištenost tablice raspršenog adresiranja. U tu svrhu iskorišten je rekurzivni model indeksa koji je modelirao funkciju distribucije ključeva koji su spremani u tablicu. Rekurzivni model indeksa je varijacija na strukturu mješavine eksperata, hijerarhijske strukture u strojnom učenju. Učenje se provodilo koristeći stohastičko učenje funkcije razdiobe.

Dobiveni rezultati pokazuju smanjeni broj kolizija u svim slučajevima odnosa veličine skupa ključeva i veličine tablice raspršenog adresiranja. Pokazani rezultati drže za 16-torke nukleotida, no korištena struktura bi se trebala moći generalizirati na k-torke proizvoljnih duljina kao i tablice proizvoljnih veličina, što je potrebno pokazati eksperimentalno. Rezultati također pokazuju vremenski omjer funkcije sažimanja i primijenjene RMI strukture u korist funkcije sažimanja, no razvojem grafičkih kartica moguće je tu razliku smanjiti u budućnosti.

Osim korištenog modela, mogući su modeli s drugačijim brojem razina i metodom učenja distribucije koji se razlikuje od implementiranog u ovom radu. Trebalo bi dalje istražiti kako drugačiji model aproksimacije distribucije utječe na raspodjelu ključeva po pretincima tablice te kako broj razina hijerarhije utječe na brzinu izvođenja.

Literatura

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, i Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd izdanju, 2009. ISBN 0262033844, 9780262033848.

[2] Michael L Fredman, János Komlós, i Endre Szemerédi. Storing a sparse table with 0 (1) worst case access time. *Journal of the ACM (JACM)*, 31(3): 538–544, 1984.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL `http://arxiv.org/abs/1502.01852`.

[4] Geoffrey Hinton i Kevin Srivastava, Nitish adn Swersky. Overview of mini-batch gradient descent, 2018. URL `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

[5] J.Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143 – 154, 1979. ISSN 0022-0000. doi: {https://doi.org/10.1016/0022-0000(79)90044-8}. URL `http://www.sciencedirect.com/science/article/pii/0022000079900448`.

[6] Luke Johnson, Moses Charikar, i G. Valiant. Cs161lecture08, 2017. URL `https://web.stanford.edu/class/cs161/Lectures/Lecture8/CS161Lecture08.pdf`.

[7] Josip Krapac. du1feedforward, 2018. URL `http://www.zemris.fer.hr/~ssegvic/du/du1feedforward.pdf`.

[8] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, i Neoklis Polyzotis. The case for learned index structures. *arXiv preprint arXiv:1712.01208*, 2017.

[9] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016.

[10] Malik Magdon-Ismail i Amir Atiya. Neural networks for density estimation. 11 1999.

[11] Kurt Mehlhorn i Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer, 2008. ISBN 978-3-540-77977-3. doi: 10.1007/978-3-540-77978-0. URL https://doi.org/10.1007/978-3-540-77978-0.

[12] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, i Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *CoRR*, abs/1701.06538, 2017. URL http://arxiv.org/abs/1701.06538.

[13] Joseph Sill i Yaser S Abu-Mostafa. Monotonicity hints. U *Advances in neural information processing systems*, stranice 634–640, 1997.

[14] J Taylor. Stp 421 lecture 18, 2010. URL https://math.la.asu.edu/~jtaylor/teaching/Fall2010/STP421/lectures/lecture18.pdf.

[15] Mikkel Thorup. High speed hashing for integers and strings. *CoRR*, abs/1504.06804, 2015. URL http://arxiv.org/abs/1504.06804.

[16] Unknown. Blast query input and database selection, 2018. URL https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=BlastHelp.

[17] Jingdong Wang, Heng Tao Shen, Jingkuan Song, i Jianqiu Ji. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014. URL http://arxiv.org/abs/1408.2927.

Naučene indeksne strukture

Sažetak

Tradicionalne strukture podataka za indeksiranje mogu se poboljšati metodama strojnog učenja. Neuronske mreže za tu primjenu uče distribuciju podataka spremljenih u indeksne strukture. Razvoj grafičkih kartica može povećati efikasnost i smanjiti vremena izvođenja takvih modela.
U ovom radu pokazano je da naučeni model može bolje raspoređivati ključeve po tablici raspršenog adresiranja smanjujuću broj kolizija i memorijsko zauzeće. U radu je predstavljena teorijska podloga tablica raspršenog adresiranja i struktura modela strojnog učenja te implementacijski detalji razvijenog programskog koda.

Ključne riječi: bioinformatika, strojno učenje, ktorka, tablica raspršenog adresiranja, rekurzivni model indeksa, neuronska mreža

Learned index structures

Abstract

Traditional index data structures can be improved using machine learning. Neural networks used for this learn the underlying data distributions of the data which is stored in these data structures. The development of graphics processing units could improve the efficiency of such models and reduce their execution time. It is shown in this thesis that a model can be learned which distributes the keys over the hash table more efficiently, thereby reducing the number of collisions as well as reducing the memory footprint. This thesis presents the theoretical background of hash tables and the machine learning structure as well as the implementation details of the developed source code.

Keywords: bioinformatics, machine learning, kmer, hash table, recursive model index, neural network