Vizualizacija struktura podataka i algoritama

Rukavina, Ivan Karlo

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split, University of Split, Faculty of science / Sveučilište u Splitu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:166:351062

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

DIPLOMSKI RAD

Vizualizacija struktura podataka i algoritama

Ivan Karlo Rukavina

Split, rujan 2022.

Temeljna dokumentacijska kartica

Diplomski rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

VIZUALIZACIJA STRUKTURA PODATAKA I ALGORITAMA

Ivan Karlo Rukavina

SAŽETAK

Cilj ovog rada je pokazati mogućnosti i prednosti korištenja vizualizacije u učenju struktura podataka i algoritama. Kroz rad ćemo se upoznati s najvažnijim strukturama podataka, te njihovom primjenom. Objasniti ćemo neke značajne algoritme za pretraživanje i sortiranje navedenih struktura. U nastavku rada objasniti ćemo ulogu vizualizacije u edukaciji, s posebnim naglaskom na vizualizaciju pri učenju struktura podataka i algoritama. Na kraju rada ćemo pokazati projekt koji smo izradili kao pomoć pri učenju struktura podataka.

Ključne riječi: Strukture podataka, algoritmi, vizualizacija

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 50 stranica, 18 grafičkih prikaza, 4 tablice i 30 literaturnih navoda.

Izvornik je na hrvatskom jeziku.

Mentor: Dr.sc. Divna Krpan, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Ocjenjivači: Dr.sc. Divna Krpan, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Dr.sc. Saša Mladenović, izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Dr.sc. Goran Zaharija, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad prihvaćen: Rujan, 2022.

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Computer Science
Ruđera Boškovića 33, 21000 Split, Croatia

Visualization of data structures and algorithms

Ivan Karlo Rukavina

ABSTRACT

This paper aims to show the possibilities and advantages of using visualization in learning data structures and algorithms. Through this paper, we will learn about the most important data structures and their application. We will explain some essential algorithms for searching and sorting the specified data structures. In the continuation of the paper, we will explain the role of visualization in education, with particular emphasis on visualization when learning data structures and algorithms. At the end of the thesis, we will show the project created to help with learning data structures.

Key Words: Data structures, algorithms, visualization

Thesis deposited in library of Faculty of science, University of Split.

Thesis consist of: 50 pages, 18 figures, 4 tables and 30 references

Original language: Croatian

Supervisor: **Divna Krpan**, **Ph.D.** Assistant Professor of Faculty of Science, University of Split

Reviewers: **Divna Krpan**, **Ph.D.** Assistant Professor of Faculty of Science, University of Split,

Saša Mladenović, Ph.D. Associate Professor of Faculty of Science, University of Split,

Goran Zaharija, Ph.D. Assistant Professor of Faculty of Science, University of Split

Thesis accepted: September, 2022.

IZJAVA

kojom izjavljujem s punom materijalnom i moralnom odgovornošću da sam diplomski/završni rad s naslovom *Vizualizacija struktura podataka i algoritama* izradio samostalno pod voditeljstvom Dr.sc. Divna Krpan, U radu sam primijenio metodologiju znanstvenoistraživačkog rada i koristio literaturu koja je navedena na kraju diplomskog rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući naveo u diplomskom radu na uobičajen, standardan način citirao/la sam i povezao s fusnotama s korištenim bibliografskim jedinicama. Rad je pisan u duhu hrvatskog jezika.

Student

Ivan Karlo Rukavina

Sadržaj:

1. Uvod

Poučavanje programiranja je zahtjevan posao. Osim što je učeniku potrebno objasniti na koji će način nešto napraviti, kako bi on mogao napredovati, ključno je razumjeti zašto je napravio određeni korak, zašto je koristio određenu tehnologiju i koje su bile alternative. Kako učenik napreduje i proširuje svoje znanje, stvarat će sve složenije aplikacije i rješavati sve teže probleme. U jednom trenutku imati će potrebu koristiti određene strukture podataka i algoritme, koje je potrebno razumjeti za ispravnu implementaciju. Razne strukture i algoritmi su teški za shvatiti početniku, ali su ključni za daljnji razvoj. Za bolje razumijevanje ove tematike, značajno nam može pomoći vizualizacija.

Cilj ovog rada je upoznati se s vizualizacijom kao izvrsnim alatom pri učenju veoma zahtjevne teme u programiranju.

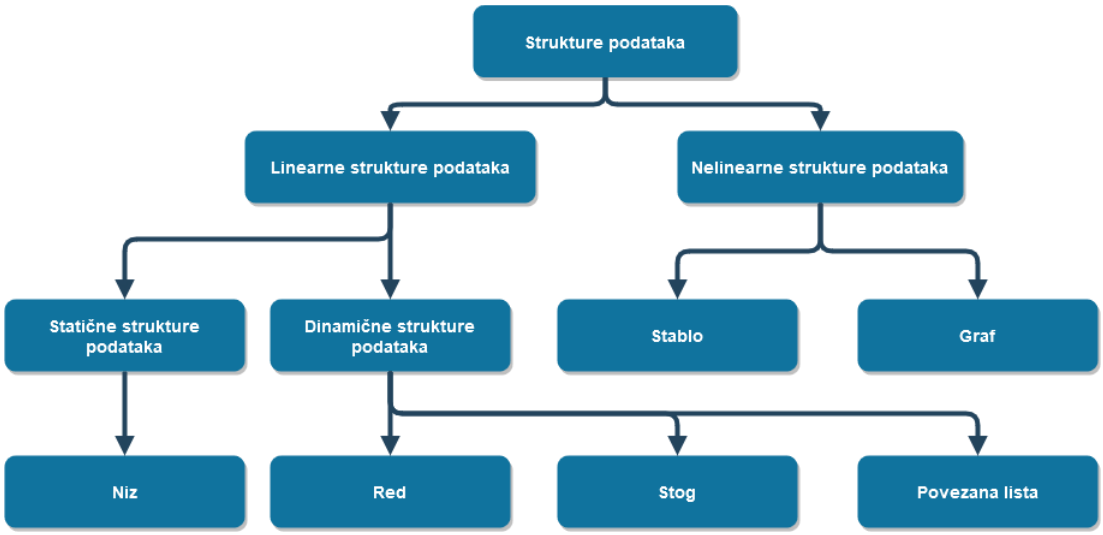
U ovome radu bavimo se vizualizacijom struktura podataka i algoritama koji se često koriste. U prvome dijelu rada objasniti ćemo najčešće strukture podataka, primjene i implementaciju. Zatim ćemo se posvetiti algoritmima, koji su usko povezani sa strukturama podataka. Objasniti ćemo što su to algoritmi, zašto su bitni i kako su povezani s strukturama podataka, te ćemo navesti neke važne i poznate algoritme. U trećem dijelu ćemo objasniti vizualizaciju, na koji je način primjenjujemo pri učenju programiranja te koje su njene prednosti, ali i mane. Na kraju ćemo napraviti prototip aplikacije za vizualizaciju struktura podataka, te opisati funkcionalnosti i proces izrade, kao i daljnje ideje za razvoj takve aplikacije.

2. Strukture podataka

Struktura podataka je način pohranjivanja i organiziranja podataka. To je način raspoređivanja podataka na računalu tako da im se može pristupiti i učinkovito ažurirati. Struktura podataka ne služi samo za organiziranje podataka. Također se koristi za obradu, dohvaćanje i pohranu podataka. Postoje različite osnovne i napredne vrste struktura podataka koje se koriste u gotovo svakom programu ili softverskom sustavu koji je razvijen. Stoga je vrlo bitno za svakog programera upoznati se i razumjeti strukture podataka. U nekim slučajevima, osnovne operacije algoritma su usko povezane s dizajnom podatkovne strukture. Svaka struktura podataka sadrži informacije o vrijednostima podataka, odnosima između podataka i u nekim slučajevima funkcijama koje se mogu primijeniti na podatke.

Strukture podataka možemo podijeliti na linearne i nelinearne. Linearna struktura podataka je ona u kojoj su elementi podataka raspoređeni sekvencijalno ili linearno, gdje je svaki element povezan sa svojim prethodnim i sljedećim susjednim elementima. One mogu biti statične, što znači da zauzimaju fiksnu količinu memorije pri svojem kreiranju ili dinamične, što znači da se tijekom izvođenja programa mijenjaju. Nelinearna struktura podataka je ona u kojoj elementi podataka nisu postavljeni sekvencijalno ili linearno. U nelinearnoj strukturi podataka, ne možemo proći sve elemente samo u jednom izvođenju. [1]

Primjeri linearnih struktura koje se često koriste su vezane liste, red i stog, dok su primjeri nelinearnih struktura stabla i grafovi. U ovom poglavlju ćemo objasniti najčešće strukture, te navesti neke od njihovih primjena u stvarnome svijetu.



Slika 1 – Klasifikacija struktura podataka

Strukture podataka koje ćemo prve prokomentirati u ovom poglavlju će biti niz i vezane liste. Zatim ćemo pogledati još dvije iznimno važne strukture koje susrećemo u svakodnevnom životu i bez kojih je ozbiljnije programiranje praktički nemoguće, a to su red (*eng. Queue*) i stog (*eng. Stack*). Nakon toga na kraju poglavlja pogledati ćemo dvije veoma važne nelinearne strukture podataka, a to su stabla (s posebnim naglaskom na binarna stabla), te grafovi.

Tipični osnovni tipovi podataka, kao što su cijeli brojevi ili vrijednosti s pomičnim zarezom (*eng. float*), koji su dostupni u većini računalnih programskih jezika općenito su nedostatni za hvatanje logičke namjere za obradu i upotrebu podataka. Ipak, aplikacije koje unose, manipuliraju i proizvode informacije moraju razumjeti kako bi podaci trebali biti organizirani da bi se pojednostavila obrada. Strukture podataka spajaju elemente podataka na logičan način i olakšavaju učinkovitu upotrebu, postojanost i dijeljenje podataka. Oni pružaju formalni model koji opisuje način na koji su elementi podataka organizirani. [2]

2.1. Niz i vezane liste

U ovom dijelu ćemo objasniti što su to nizovi, liste i vezane liste te gdje se koriste. U stvarnome svijetu često imamo potrebu određene objekte, pojave i slično grupirati i/ili poredati. U programiranju je cilj neki problem iz stvarnog života riješiti pomoću programskog jezika, tako da ćemo često, ako ne i uvijek imati potrebu spremiti određen broj podataka koji će nam kasnije trebati. Tu je iznimno važno da smo organizirani kako bi kasnije mogli lako pristupiti tom podatku. Recimo na primjer da naš prijatelj ima liječničku ordinaciju koju želi digitalizirati. Do sad je sve pacijente držao u velikoj ladici, u kojoj je imao preko 500 "kartona" poredanih po abecednom redu. Krećemo programirati i prije nego napišemo prvu liniju koda trebamo riješiti jedan bitan problem, na koji način ćemo spremiti pacijente? Jasno je da nam je potrebna neka struktura koja će omogućiti jednostavno pretragu, ali i mogućnost izmjene i brisanja.

Tu dolazimo do prve "grupe" struktura podataka koje ćemo u ovom radu objasniti, a tu su niz, lista i vezana lista. Treba napomenuti da se nizovi i liste, te njihove mogućnosti djelomično razlikuju s obzirom koji programski jezik koristimo, ali ideja je ista kod svih programskih jezika, o ovome primjeru pričati ćemo za nizove i liste u programskom jeziku Python, koji je čest izbor pri učenju programiranja. Prvo što moramo razjasniti je razlika između niza i liste, koja često može zbuniti nekoga tko prvi put uči strukture podataka, jer su jako slične. Sličnosti niza i liste su da se oboje koriste za spremanje podataka, obje su promjenjive, te se mogu indeksirati.

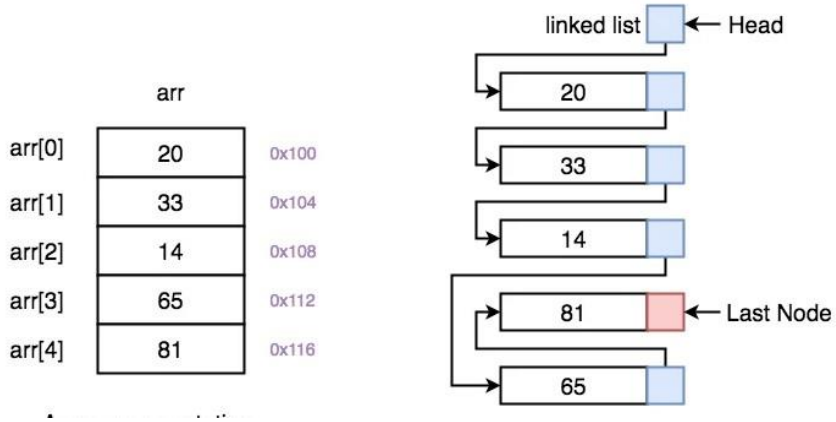
Glavna razlika između ove dvije vrste podataka je operacija koju možete izvesti na njima. Nizovi su posebno optimizirani za aritmetičke izračune, pa ako se izvode slične operacije, korištenje niza je vjerojatno bolje od liste. Liste su također spremnici za elemente koji imaju različite tipove podataka, ali nizovi se koriste kao spremnici za elemente istog tipa podataka. [3]

U sljedećoj tablici smo naveli usporedbu niza i vezane liste.

NIZ	VEZANA LISTA
Fiksirana veličina, promjena je “skupa”	Dinamična veličina
Umetanje i brisanje elemenata nije efikasno, elementi se obično premještanju (*eng. shifting*)	Umetanje i brisanje elemenata je efikasno
Nasumični pristup, efektivno indeksiranje	Nema nasumičnog pristupa, nije prilagođena za operacije pristupa elemenata po indeksu
Ne zauzima bespotrebnu memoriju ako je niz pun ili skoro pun, inače postoji mogućnost da niz bespotrebno zauzme previše memorije	S obzirom da se memorija dodjeljuje dinamično, s obzirom na naše potrebe, nema bespotrebnog zauzimanja memorije
Sekvencijalni pristup je brži, jer su elementi spremljeni na susjednim memorijskim lokacijama	Sekvencijalni pristup je sporiji, jer elementi nisu spremljeni na susjednim memorijskim lokacijama

Tablica 1 – Usporedba niza i vezane liste

Pogledajmo sad sliku koja nam prikazuje kako se podaci spremaju u memoriji kod niza, a kako kod vezane liste. [5]



Slika 2 – Usporedba niza i vezane liste u memoriji, preuzeto iz [25]

Analizirajmo *sliku 2.* S lijeve strane vidimo niz, adrese niza su dodijeljene sekvencijalno, svaki element niza zauzima točno određenu veličini, te iza njega ide idući element. Kao što smo i naveli u *tablici 1* ovo je jedna od prednosti niza, kod velikih količina podataka gdje radimo neke matematičke operacija nad nizom, niz će rezultirati bržim performansama.

Vezana lista je napravljena tako da svaki element osim što sadrži neki podatak, sadrži i "pointer". Pokazivač (*eng. Pointer*) pokazuje na sljedeći element i bez njega bi podaci u vezanoj listi bili izgubljeni u memoriji. Pokazivač nam u biti služi za vezu između naših elemenat. Kod vezane liste na našoj slici imamo još dva pojma, to su *head* i *last node.* Head pokazuje na početak vezane liste, a Last node na kraj liste.

Navedimo sad neke primjere u kojima koristimo vezanu listu:

- Implementacija nizova i redova
- Implementacija grafova: Predstavljanje grafova popisom susjedstva je najpopularniji način, koji koristi vezanu listu za pohranjivanje susjednih vrhova.
- Dinamička dodjela memorije: koristimo vezanu listu slobodnih blokova. Održavanje imenika imena
- Izvođenje aritmetičkih operacija nad dugim cijelim brojevima
- Manipulacija polinomima pohranjivanjem konstanti u čvoru vezane liste koja Predstavlja rijetke matrice [6]

Možemo još napomenuti da su dostupne su 3 različite implementacije vezane liste, a to su:

- Pojedinačno vezana lista
- Dvostruko vezana lista
- Kružna vezana lista

2.2. Red i stog

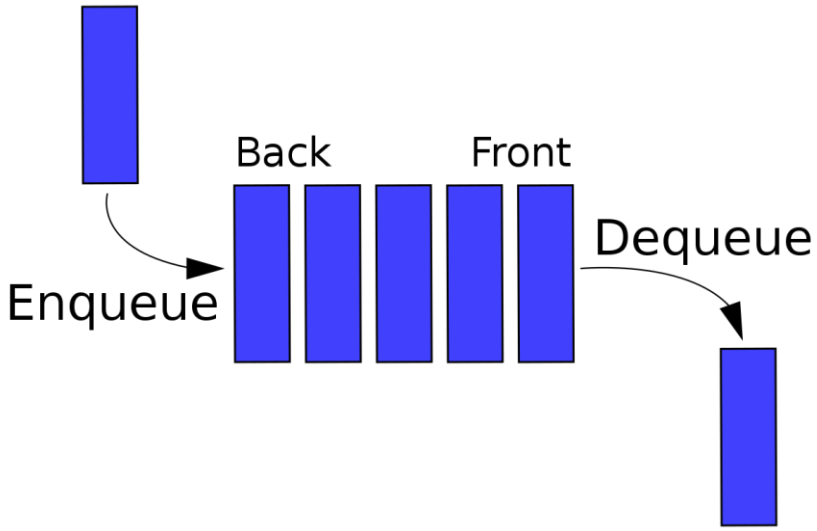
Drugi "par" linearnih struktura podataka koje ćemo obraditi biti će red i stog. Red (*eng. queue*) i stog (*eng. stack*) su dvije veoma važne strukture, s kojima se svatko od nas susreo još dok smo bili djeca. Sjetimo se samo kad smo trebali čekati red za kruh, ili smo igrali na karte te izvlačili kartu po kartu. Red i stog u programiranju se ponašaju identično.

S obzirom da s programiranjem zapravo rješavamo neke probleme iz stvarnoga svijeta, sasvim je očekivano da će se ove dvije strukture često koristiti.

Prvi od ova dva pojma koja ćemo objasniti je red. Recimo da nam je istekla osobna iskaznica i trebamo otići do policijske postaje izvaditi novu. Vrlo je vjerojatno da će kad dođemo tamo postojati određeni red. Svaka osoba je pri dolasku uzela svoj redni broj, stala u red i čekala. Logično, osoba koja je prije stala u red prije će doći na vrh reda i obaviti svoj zadatak, tj. u našem slučaju predate zahtjev za novu osobnu, a potom će napustiti red. Osoba koja je došla zadnja, tj. u ovom slučaju mi, stati će na kraj reda.

Slika 3 – Primjer reda u stvarnome životu

Možemo primijetiti da red funkcionira po principu tko prvi dođe, prvi izađe tj. "**FIFO**" metodologija (*eng. First In First Out).* Isti način će biti kod implementiranja reda u programiranju, element koji prvi dodamo u red prvi će "nešto obaviti" ili ćemo na njemu napraviti neku operaciju.



Slika 4 – struktura reda s osnovnim operacijama, preuzeto iz [26]

Red je otvoren na oba kraja. Jedan kraj se uvijek koristi za umetanje podataka čelo reda (*eng. enqueue),* a drugi, tj. Ukloni s kraja (*eng. dequeue*). koristi se za uklanjanje podataka.

Navedimo neke tipične operacije koje imamo kod reda:

- **enqueue(element)** – dodaj element na kraj reda
- **dequeue()** – ukloni element s početka reda
- **isEmpty()** – provjeri da li je red prazan
- **isFull()** – provjeri da li je red pun
- **peek()** – vrati element na vrhu reda

Sada imamo bolje razumijevanje šta je to red, pitanje koje se postavlja je sljedeće: kako ga implementirati?

Prva opcija koja nam je na raspolaganju je niz. Definirati ćemo duljinu niza, i spremati ćemo podatke u niz. Nakon dodavanja svakog novog podatka morati ćemo pomaknuti niz za 1 mjesto, tj. "shiftat" ćemo ga, ali kako već rekli, niz struktura koja jako dobro obavlja "shift". Također pošto je niz statične veličine, trebamo paziti da ne pokušamo dodati novi element ali je niz pun, tj. *niz[lastIndex]* treba biti manji ili jednak *n-1*, gdje je *n* maksimalan broj elemenata niza, a *lastIndex* je indeks zadnjeg elementa u nizu.

Prednost ovog pristupa je:

- Jednostavan za implementaciju.
- S lakoćom se može učinkovito upravljati velikom količinom podataka.
- Operacije kao što su umetanje i brisanje mogu se izvesti s lakoćom jer slijede pravilo prvi ušao prvi izašao.

Sljedeća opcija za implementirati red je vezana lista. Prednost vezane liste je što je dinamična, a biti će nam potrebna dva pointera koji će pokazivati na početak i kraj reda (Rear pointer i Front pointer). Dodavanjem novog elementa samo prebacimo Rear pointer na taj novi element, a kad izbacimo prvi element s vezane liste prebaciti ćemo Front pointer na sljedeći element. Vremenska složenost obje operacije enqueue() i dequeue() je O(1) jer mijenjamo samo nekoliko pokazivača u obje operacije. Nema petlje ni u jednoj operaciji. Prostorna složenost obje operacije enqueue() i dequeue() je O(1) jer je potreban stalni dodatni prostor. [7]

Ostaje nam još objasniti kružni red. Kružni red je posebna verzija reda čekanja gdje je posljednji element reda čekanja povezan s prvim elementom reda tvoreći krug.

Primjeri korištenja su:

- Upravljanje memorijom: Neiskorištene memorijske lokacije u slučaju običnih redova čekanja mogu se koristiti u kružnim redovima čekanja.
- Prometni sustav: Kružni redovi se koriste za paljenje semafora jedan po jedan više puta prema postavljenom vremenu.
- Raspored CPU-a: Operativni sustavi često održavaju red čekanja procesa koji su spremni za izvršenje ili koji čekaju da se dogodi određeni događaj.

Jako slična struktura redu je stog. Stog je linearna struktura podataka koja slijedi određeni redoslijed u kojem se operacije izvode. Stog funkcionira po principu tko zadnji uđe, prvi izađe tj. "**LIFO**" metodologija (*eng. Last In First Out).*

Slika 5 – primjer stoga - knjige

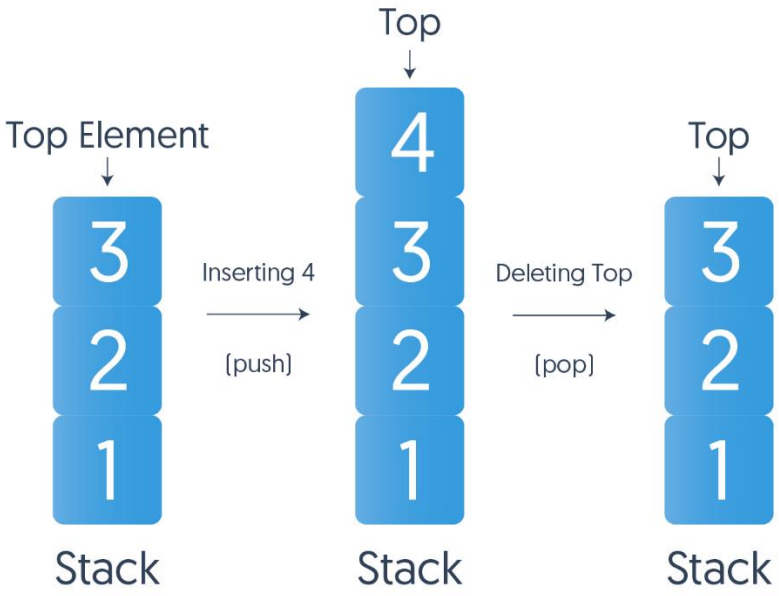
U stvarnome životu imamo puno primjera stoga, po čemu je i dobio ime. Knjige, tanjuri, pisma na stalu itd. Recimo da imamo hrpu knjiga složenih kao na *slici 5* a da ima ime piše samo na naslovnoj strani. Najlogičniji način za doći do određene knjige je uzimati jednu po jednu s vrha i provjeriti naziv, dok ne dođemo do one koja nam je potrebna. Istu logiku ćemo primijeniti pri implementiranju stoga u programiranju.

Stog i red imaju jako puno toga zajedničkog, ključna razlika je upravo na koji kraj dodajemo novi element i na kojem kraju ga uklanjamo. Stog je za razliku od reda otvoren samo na jednom kraju, na isti kraj ćemo i uklanjati elemente i dodavati ih.

Postoje dvije ključne operacije s stogom, a to su:

- **Push()** - Guranje (spremanje) novog elementa na stog.
- **Pop()** - Uklanjanje (ili pristup) elementu sa stoga.

Također imamo i sve dodatne operacije koje imamo i kod reda, a tu su **isfull(), isEmpty()** i **peek().**



Slika 6 – struktura stoga i ključni pojmovi, preuzeto iz [27]

Pogledajmo sad primjer s *slike 6.* Imamo stog s tri elementa, **1, 2, 3**. Radimo operaciju **push(4),** tj. na vrh (*eng. Head*) stoga.

Nakon te operacije imati ćemo četri elementa u stogu, a pointer od Head-a će umjesto na **3** pokazivati na element **4**. Sada u drugom koraku radimo operaciju **pop()** koja će ukloniti element koji je na vrhu stoga, a pointer od Head-a će se pomaknuti na idući element. Nakon ove operacije naš stog će opet imati tri elementa s kojima smo počeli.

Stog se može jednostavno implementirati putem niza ili vezane liste, jer su stogovi samo posebni slučajevi lista. [8]

S ovim smo obradili najčešće i najbitnije linearne strukture podataka, niz, vezane liste, red i stog. Strukture koje smo nabrojali koristiti ćemo u praktički svakom projektu, u većoj ili manjoj mjeri i ključno je dobro razumjevanje njihovih prednosti i mana.

U sljedećem dijelu ćemo se baviti s nelinearnim strukturama, tj. s stablima i grafovima.

2.3. Stabla

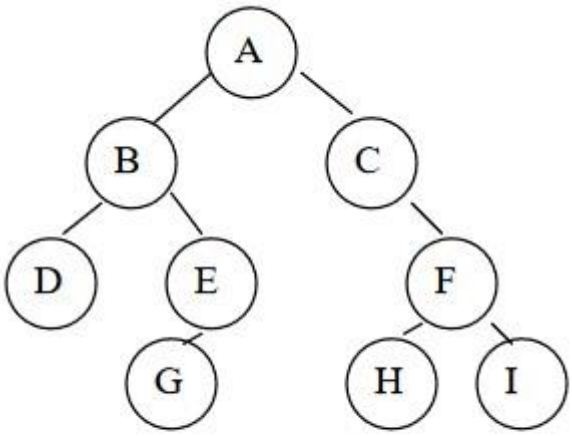
Do sad smo se upoznali s linearnim strukturama podataka. Kaže se da je struktura podataka linearna ako se njezini elementi kombiniraju kako bi formirali bilo koji specifični poredak.

Do kraja ovog poglavlja baviti ćemo se s nelineranim strukturama podataka, tj. s dvije najvažnije strukture, a to su stabla i grafovi. Ova struktura se uglavnom koristi za predstavljanje podataka koji sadrže hijerarhijski odnos između različitih elemenata. [9]

Stabla	Grafovi
U ovom slučaju podaci često sadrže hijerarhijski odnos između različitih elementi. Struktura podataka koja odražava ovaj odnos naziva se ukorijenjeno stablo grafa (*eng. rooted tree graph*) ili stablo.	U ovom slučaju, podaci ponekad imaju odnos između parova elemenata što ne mora nužno slijediti hijerarhijsku strukturu. Takva struktura podataka je naziva se graf.

Tablica 2 – usporedba stabla i grafa

Stabla predstavljaju poseban slučaj općenitijih struktura poznatih kao grafovi. U grafu nema ograničenja na broj veza koje mogu ući ili izaći iz čvora, a ciklusi mogu biti prisutni u graf. Posebno ćemo promatrati binarna stabla koja se koriste umjesto jednostavnijih struktura kao što su nizovi i vezane liste zbog bržeg pretraživanja i umetanja podataka.



Slika 7 – Primjeri stabla

Pogledajmo stablo i objasnimo neke osnovne pojmove vezano uz njega.

Svako stablo je ujedno i graf.

Stablo je konačan skup jednog ili više čvorova koji imaju jedan poseban čvor koji se zove **korijen**, a ostali su podijeljeni u nula ili više disjunktnih skupova, koji se još nazivaju **podstabla**. Tako je na primjer u našem stablu imamo podstablo kojemu je korijen B, a sadrži još čvorove D, E, G.

Čvorovi na koje pokazuje neki čvor u stablu se zovu njegovom **djecom**, a on sam njihovim **roditeljem**.

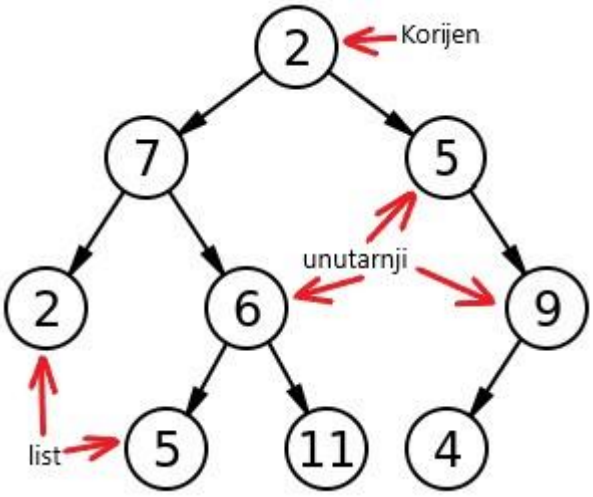
Na našem primjeru sa slike 7, čvor A je korijen stabla, to je jedini čvor koji nema roditelja. Čvor B je dijete od A, a samim time kažemo da je A roditelj od B.

Putanja (*eng. path*) je put koji prolazimo od korijena do ciljanog čvora, na primjer, za pronaći put do H, putanja će biti A-C-F-H.

U ovome slučaju naše stablo je ujedno i binarno stablo. **Binarno stablo** je stablo u kojemu svaki čvor ima nula, jedno ili dvoje djece, lijevo i desno dijete je različito. Još jedna razlika u odnosu na obično stablo je to da binarno stablo može biti prazno.

2.3.1. Binarna stabla

Objasnimo sada malo pobliže binarno stablo. U informatici, binarno stablo je struktura podataka stabla u kojoj svaki čvor ima najviše dva djeteta, koji se nazivaju lijevo i desno dijete. [10]



Slika 8 – Elementi binarnog stabla

Na *slici 8* vidimo da kao i u običnome stablu, imamo jedan korijen, u našem slučaju je to broj 8. Čvor koji nema ni jedno dijete zove se list (u našem primjeru to su 2, 11 i 4), a ako čvor nije ni list ni korijen kažemo da te to unutarnji čvor (brojevi 7, 6, 9).

Još jedna bitna podvrsta binarnogstabla koja nam je posebno zanimljiva je **uređeno binarno stablo**. Za binarno stablo kažemo da je uređeno ukoliko je svaki njegov čvor vrijedi da mu je u lijevoj grani dijete koje je manje od njega, a u desnoj dijete koje je veće od njega.

Putanja u uređenom binarnom stablu je skup čvorova počevši od korijena do nekog lista. Duljina putanje je razina lista u toj putanji + 1. Dubina (visina) stabla je duljina najdulje putanje Za uređeno binarno stablo kažemo da je balansirano ako se, za svaki čvor u njemu, dubina lijevog i desnog podstabla razlikuje za najviše 1.

2.3.2. Primjene binarnog stabla

Sad kad imamo bolju sliku šta je to binarno stablo i kako se definira, pitanje koje možemo postaviti je, zašto je ono bitno? Zašto je binarno stablo tako posebno?

Navedimo neke primjene binarnog stabla:

- Tablice usmjeravanja
- Stabla odlučivanja
- Evaluacija izraza
- Pretraživanje
- Sortiranje
- Indeksi za baze podataka
- Kompresija podataka

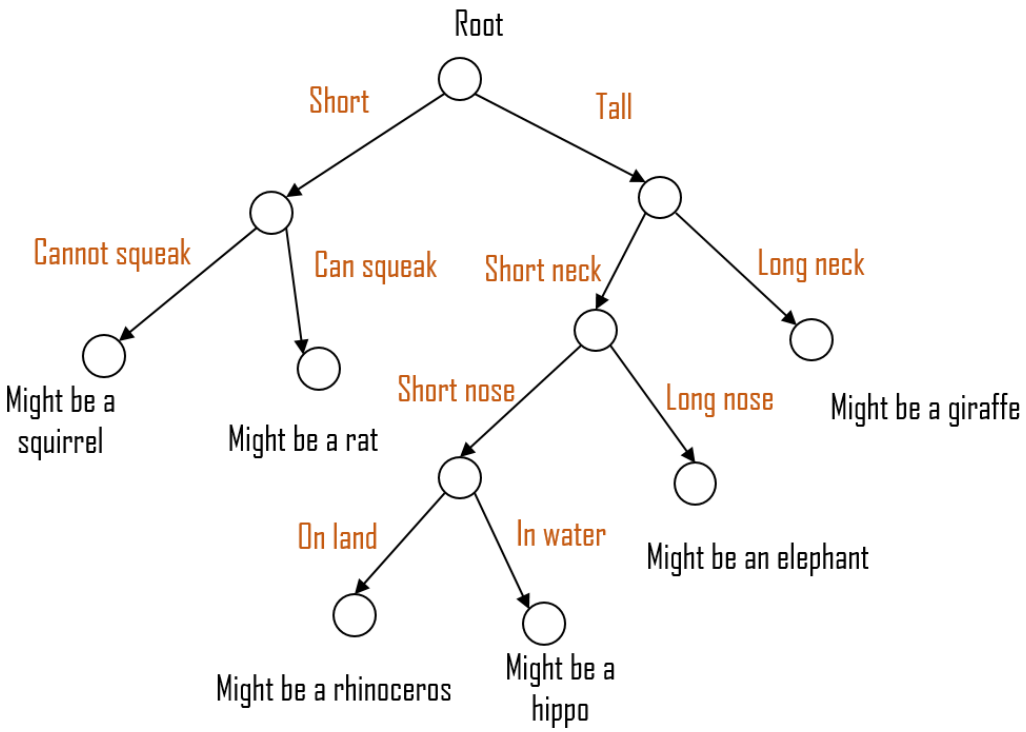
U računarstvu, binarna stabla se uglavnom koriste za pretraživanje i sortiranje jer pružaju sredstva za hijerarhijsku pohranu podataka. Neke uobičajene operacije koje se mogu provesti na binarnim stablima uključuju umetanje, brisanje i obilazak. Korištenje binarnih stabla je često najbrži način pronalaska i uređivanja nekih podataka.

Tablica usmjeravanja koristi se za povezivanje usmjerivača u mreži. Obično se implementira s podatkovnom strukturom "trie", koja je varijacija binarnog stabla. Struktura podataka stabla pohranit će lokaciju usmjerivača na temelju njihovih IP adresa. Usmjerivači sa sličnim adresama grupirani su pod jedno podstablo.

Da bismo pronašli usmjerivač na koji se paket mora proslijediti, moramo proći kroz stablo koristeći prefiks mrežne adrese na koju se paket mora poslati. Nakon toga, paket se prosljeđuje usmjerivaču s najduljim odgovarajućim prefiksom odredišne adrese.

Binarna stabla također se mogu koristiti za potrebe klasifikacije. Stablo odlučivanja je nadzirani algoritam strojnog učenja. Struktura podataka binarnog stabla ovdje se koristi za oponašanje procesa donošenja odluka. Stablo odlučivanja obično počinje korijenskim čvorom. Interni čvorovi su uvjeti ili značajke skupa podataka. Grane su

pravila odlučivanja, dok su napuštajući čvorovi ishodi odluke. *Slika 9.* je primjer stabla odlučivanja.



Slika 9 – Primjer binarnog stabla kod strojnog učenja, preuzeto iz [28]

Druga korisna primjena binarnih stabala je evaluacija izraza. U matematici, izrazi su iskazi s operatorima i operandima koji procjenjuju vrijednost. Listovi binarnog stabla su operandi dok su unutarnji čvorovi operatori. Izraz se procjenjuje primjenom operatora u internom čvoru na operande u listovima.

Binarna stabla pretraživanja (*eng. Binary search trees*), varijanta binarnih stabala koriste se u implementaciji algoritama sortiranja za poredak stavki. Binarno stablo pretraživanja jednostavno je uređeno ili sortirano binarno stablo tako da je vrijednost u lijevom podređenom čvoru manja od vrijednosti u nadređenom čvoru. U isto vrijeme, vrijednosti u desnom čvoru veće su od vrijednosti u nadređenom čvoru. Da bi se dovršio postupak sortiranja, stavke koje treba sortirati prvo se umeću u binarno stablo pretraživanja. Da bi se dohvatile sortirane stavke, stablo se obilazi korištenjem obilaska redom.

U indeksiranju baze podataka, B-stabla se koriste za sortiranje podataka za pojednostavljeno pretraživanje, umetanje i brisanje. Važno je napomenuti da B-stablo nije binarno stablo, ali to može postati kada preuzme svojstva binarnog stabla. Baza podataka stvara indekse za svaki dati zapis u bazi podataka. B-stablo zatim u svojim unutarnjim čvorovima pohranjuje reference na zapise podataka sa stvarnim zapisima podataka u svojim lisnim čvorovima. To omogućuje sekvencijalni pristup podacima u bazama podataka.

U sažimanju podataka, Huffmanovo kodiranje se koristi za stvaranje binarnog stabla sposobnog za sažimanje podataka. Kompresija podataka je obrada kodiranih podataka za korištenje manje bitova. S obzirom na tekst za komprimiranje, Huffmanovo kodiranje gradi binarno stablo i umeće kodiranje znakova u čvorove na temelju njihove učestalosti u tekstu. Kodiranje za znak dobiva se prelaskom stabla od njegovog korijena do čvora. Znakovi koji se često pojavljuju imat će kraći put u usporedbi sa znakovima koji se rjeđe pojavljuju. Ovo se radi kako bi se smanjio broj bitova za česte znakove i osigurala maksimalna kompresija podataka. [11]

Kao što vidimo, binarna stabla imaju široku primjenu u svijetu informatike, te su najčešća nelinearna struktura podataka koju koristimo.

Binarna stabla nude mnoge prednosti zbog čega ostaju vrlo korisna struktura podataka. Mogu se koristiti za prikaz strukturnih odnosa i hijerarhija u skupu podataka. Još važnije, binarna stabla omogućuju učinkovito pretraživanje, brisanje i umetanje.

Također je vrlo jednostavno implementirati i održavati binarno stablo. Binarno stablo programerima nudi prednosti uređenog niza i vezane liste; pretraživanje u binarnom stablu jednako je brzo kao u sortiranom nizu, a operacije umetanja ili brisanja jednako su učinkovite kao u vezanoj listi. [12] S binarnim stablima dobiti ćemo najbolje od oba svijeta.

2.4. Grafovi

Dolazimo do zadnje strukture podataka koju ćemo obraditi u ovom radu, a to su grafovi. Graf je nelinearna struktura podataka koja se sastoji od čvorova ili vrhova (oznaka V) i bridova (oznaka E). Rubovi povezuju bilo koja dva čvora u grafu, a čvorovi su također poznati kao vrhovi (*eng. vertices*). Oznaka za graf u matematici je G(V, E). Grafovi se koriste za rješavanje mnogih problema iz stvarnog života. Grafovi se koriste za predstavljanje mreža. Mreže mogu uključivati staze u gradskoj ili telefonskoj mreži ili mreži krugova.

Grafovi se također koriste u društvenim mrežama kao što su LinkedIn, Facebook. Na primjer, na Facebooku je svaka osoba predstavljena vrhom (ili čvorom). Svaki čvor je struktura i sadrži informacije kao što su ID osobe, ime, spol, lokalicija itd.

Stabla su ograničene vrste grafova, samo s još nekim pravilima. Svako će stablo uvijek biti graf, ali neće svi grafovi biti stabla. Vezane liste, stabla i gomile posebni su slučajevi grafova. [13]

Postoje dva načina za pohranjivanje grafikona:

- Matrica susjedstva
- Popis susjedstva

Matrica susjedstva - u ovoj metodi, graf je pohranjen u obliku 2D matrice gdje redovi i stupci označavaju vrhove. Svaki unos u matricu predstavlja težinu brida između tih vrhova.

Popis susjedstva - ovaj je grafikon predstavljen kao zbirka vezanih lista. Postoji niz pokazivača koji pokazuje na rubove povezane s tim vrhom.

Osnovne operacije na grafu su:

- Umetanje čvorova/rubova u graf – umetanje čvora u graf.
- Brisanje čvorova/rubova u grafu – Izbrišite čvor iz grafa.
- Pretraživanje na grafikonima – pretražite entitet na grafikonu.
- Obilaženje grafova – Obilaženje svih čvorova u grafu.

Usporedimo sad graf i stablo, koje su dvije nelinearne strukture podataka.

Osnova za usporedbu	Graf	Stablo
Definicija	Graf je nelinearna struktura podataka.	Stablo je nelinearna struktura podataka.
Struktura	To je skup vrhova/čvorova i bridova.	To je skup čvorova i bridova.
Bridovi	Svaki čvor može imati proizvoljan broj bridova.	Ako postoji n čvorova tada bi postojao n-1 broj rubova.
Vrsta bridova	Mogu biti usmjereni i neusmjereni.	Uvijek su usmjereni.
Korijen	U grafu ne postoji jedinstveni čvor koji se zove korijen.	U stablima postoji jedinstveni čvor koji se naziva korijenski (roditeljski) čvor.
Petlja	Može se formirati ciklus.	Neće biti nikakvog ciklusa.
Šetnja	Za obilazak grafa koristimo u širinu *(eng. Breadth-First Search - BFS*) i pretraživanje u dubinu (*eng. Depth-First Search - DFS*).	Prolazimo kroz stablo korištenjem metoda obilaženja prema redoslijedu, unaprijed ili nakon redoslijeda. (*eng. in-order, pre-order, post-order*)
Primjena	Za pronalaženje najkraćeg puta u mrežnom grafu koristi se graf.	Za stabla igre, stabla odlučivanja, koristi se stablo.

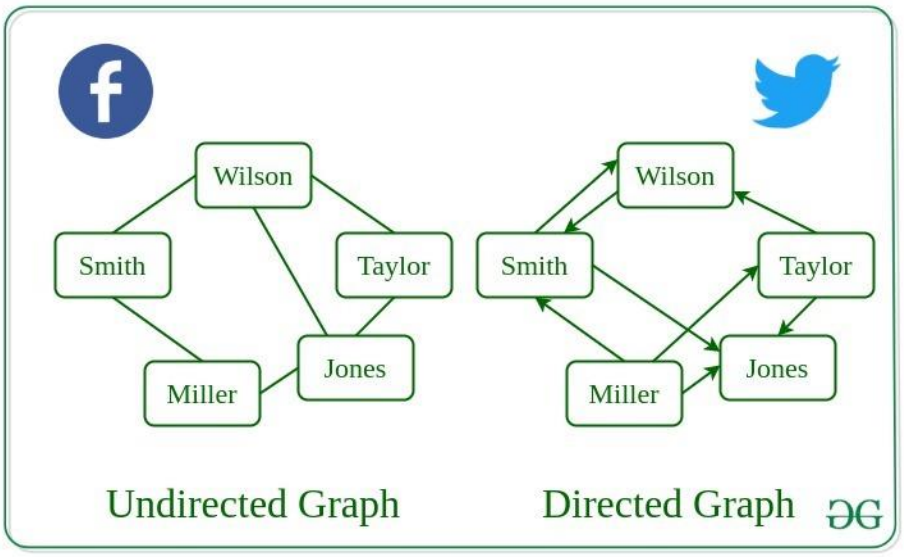
Tablica 3 – usporedba grafa i stabla

Primjene grafova u stvarnom životu:

- Karte se mogu prikazati pomoću grafikona, a zatim ih mogu koristiti računala za pružanje raznih usluga poput najkraćeg puta između dva grada.
- Kada različiti zadaci ovise jedni o drugima, tada se ova situacija može prikazati korištenjem usmjerenog acikličkog grafa i možemo pronaći redoslijed kojim se zadaci mogu izvršiti pomoću topološkog sortiranja.
- Dijagram prijelaza stanja predstavlja što mogu biti legalni potezi iz trenutnih stanja. Ovo se može koristiti u igri Križić-kružić (*eng. tic tac toe*). [13]

Vrste grafova:

1. Konačni grafovi - za graf se kaže da je konačan ako ima konačan broj vrhova i konačan broj bridova.
2. Beskonačni graf - za graf se kaže da je beskonačan ako ima beskonačan broj vrhova kao i beskonačan broj bridova.
3. Trivijalni graf: Kaže se da je graf trivijalan ako konačan graf sadrži samo jedan vrh i nema ruba.
4. Jednostavan graf: Jednostavan graf je graf koji ne sadrži više od jednog brida između para vrhova. Jednostavna željeznička pruga koja povezuje različite gradove primjer je jednostavnog grafikona.
5. Višestruki graf: Svaki graf koji sadrži neke paralelne bridove, ali ne sadrži samo petlju naziva se multigraf.
6. Nulti graf: Graf reda *n* i veličine nula je graf u kojem postoje samo izolirani vrhovi bez bridova koji povezuju bilo koji par vrhova.
7. Potpuni graf - jednostavan graf s *n* vrhova naziva se potpunim grafom ako je stupanj svakog vrha *n-1*, odnosno, jedan vrh je spojen s *n-1* bridova ili ostatak vrhova u grafu. Potpuni graf se još naziva i puni graf.
8. Jednostavan graf - za graf kažemo da je regularan ako su svi vrhovi grafa *G* jednakog stupnja. Svi potpuni grafovi su pravilni, ali obrnuto nije moguće.



Slika 10 – usporedba usmjerenog i neusmjerenog grafa, preuzeto iz [29]

Postoji još raznih vrsta i podjela ali ovu su neke osnovne i najčešće s kojima ćemo se susretati. Kao što vidimo grafovi su iznimno važni, imaju široku primjenu i postoji cijelo područje u matematici koje se bavi grafovima, **Teorija grafova.** Za kraj ćemo navesti neke prednosti i mane korištenja grafova:

Prednosti grafova:

- Korištenjem grafova možemo lako pronaći najkraći put, susjede čvorova i još mnogo toga.
- Grafovi se koriste za implementaciju algoritama kao što su DFS i BFS.
- Pomaže u organiziranju podataka.
- Zbog svoje nelinearne strukture pomaže u razumijevanju složenih problema i njihovoj vizualizaciji.

Mane grafova:

- Grafovi koriste puno pokazivača s kojima može biti složeno rukovati.
- Može imati veliku memorijsku složenost.
- Ako je graf predstavljen matricom susjedstva, tada ne dopušta paralelne bridove i množenje grafa je također teško.

3. Algoritmi

3.1. Algoritmi u strukturama podataka

Algoritam je riječ koju čujemo jako često u programiranju. Algoritam za određeni zadatak može se definirati kao “konačan niz instrukcija, od kojih svaka ima jasno značenje i može se izvesti s konačnom količinom napora u konačnom vremenu".[14] Kao takav, algoritam mora biti dovoljno precizan da ga ljudi razumiju.

Međutim, da bi ga računalo izvršilo, općenito će nam trebati program koji je napisan rigoroznim formalnim jezikom, a budući da su računala prilično nefleksibilna u usporedbi s ljudskim umom, programi obično trebaju sadržavati više detalja od algoritama. Algoritam možemo shvatiti kao upute kako nešto napraviti. Na primjer, algoritam za kuhanje jaja bi mogao glasiti: Uzmi lonac, napuni lonac vodom do polovice, uključi štednjak na srednju razinu, stavi lonac s vodom na štednjak, ubaci jaje, kuhaj ga tri minute, isključi štednjak, ukloni lonac i izvadi jaje. Ovaj algoritam narovno može imati još više detalja, ali ideja je da su to “upute” kako nešto izvesti.

Algoritmi se očito mogu opisati na jednostavnom hrvatskom, engleskom i slično, a mi ćemo to ponekad i učiniti. Međutim, informatičarima je obično lakše i jasnije koristiti nešto što dolazi negdje između formatiranog engleskog i računalnog programskog koda, ali se ne može izvoditi jer su izostavljeni pojedini detalji. To se zove **pseudokod**, koji dolazi u različitim verzijama oblicima. Često će ove bilješke predstavljati segmente pseudokoda koji su vrlo slični jezika koji nas uglavnom zanimaju.

Važno je napomenuti da su algoritmi i strukture podataka neodvojivi, jedno bez drugoga nema smisla, zato ih i najčešće učimo zajedno. Strukture podataka nam omogućuju spremati podatke s kojima radimo, dok algoritmi omogućuju manipulaciji tim podacima.

Sa stajališta strukture podataka, nabrojimo neke važne kategorije algoritama:

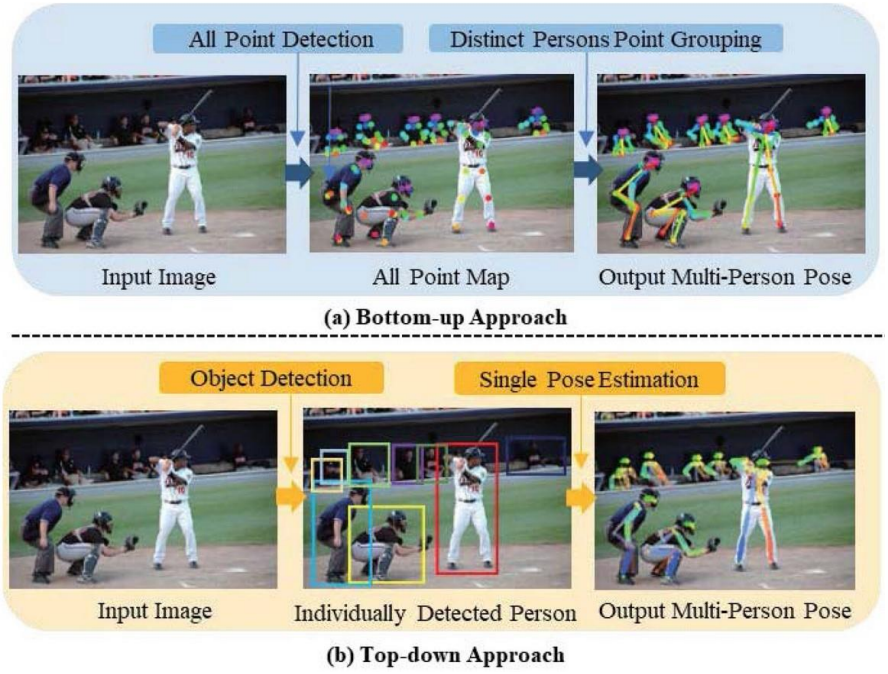
- Pretraživanje − Algoritam za pretraživanje stavke u strukturi podataka.
- Sortiranje − Algoritam za sortiranje stavki određenim redoslijedom.
- Umetanje − Algoritam za umetanje stavke u strukturu podataka.
- Ažuriranje − Algoritam za ažuriranje postojeće stavke u strukturi podataka.
- Brisanje − Algoritam za brisanje postojeće stavke iz strukture podataka.

Ne mogu se sve procedure nazvati algoritmima. **Algoritam bi trebao imati sljedeće karakteristike**:

- **Jasan i nedvosmislen**: Algoritam treba biti jasan i nedvosmislen. Svaki od njegovi koraci trebaju biti jasni u svim aspektima i moraju voditi samo jednom značenju.
- **Dobro definirani ulazi**: Ako algoritam kaže da treba uzeti ulaze, trebao bi biti dobro definirani ulazi.
- **Dobro definirani izlazi**: algoritam mora jasno definirati koji će biti izlaz donio i također treba biti dobro definiran.
- **Konačan**: algoritam mora biti konačan, tj. ne bi trebao završiti u beskonačnom petlje ili slično.
- **Izvediv**: Algoritam mora biti jednostavan, generički i praktičan, takav da može izvršiti ovisno o raspoloživim resursima. Ne smije sadržavati neku budućnost tehnologija ili bilo što.
- **Neovisan o jeziku**: dizajnirani algoritam mora biti jezično neovisno, tj. to moraju biti samo obične upute koje se mogu implementirati u bilo koji jeziku, a rezultat će biti isti, kao što se i očekivalo. [15]

Kod izrade algoritma postoje dva pristupa, pristup "odozgo prema dolje" (*eng. Top-Down Approach*) i pristup "odozdo prema gore" (*eng. Bottom-Up Approach*).

- **Pristup odozgo prema dolje**: Pristup odozgo prema dolje počinje identificiranjem glavnih komponente sustava ili programa razlažući ih na njihove komponente niže razine & ponavljanje dok se ne postigne željena razina složenosti modula. U ovome počinjemo najviši modul & postupno dodajte module koji su pozivi.
- **Pristup odozdo prema gore**: Pristup odozdo prema gore počinje projektiranjem najosnovnijih ili primitivna komponenta i prelazi na komponente više razine. Počevši od samog dna, implementiraju se operacije koje daju sloj apstrakcije.



Slika 11 – usporedba Bottom-Up i Top-Down pristupa, preuzeto iz [16]

Pogledajmo *sliku 11*. Kod Bottom-Up pristupa, kad analiziramo sliku, prvo ćemo mapirati sve točke na slici, a zatim ćemo prepoznati poze u kojima su ljudi. [16]

Kod Top-down pristupa, prvo detektiramo pojedinačne ljude, a onda prepoznajemo poze u kojima se oni nalaze.

U nastavku ćemo navesti neke od najbitnijih algoritama koji su usko vezani uz strukture podataka koje smo u ovome radu obradili.

3.2. Vremenska i prostorna složenost algoritma

Općenito, uvijek postoji više od jednog načina za rješavanje problema u informatici s različitim algoritmima. Stoga je vrlo potrebno koristiti metodu za usporedbu rješenja kako bi se procijenilo koje je optimalnije.

Metoda mora biti:

- Neovisno o stroju i njegovoj konfiguraciji na kojoj se algoritam izvodi.
- Pokazuje izravnu korelaciju s brojem ulaza.
- Može jasno razlikovati dva algoritma bez dvosmislenosti.

Postoje dvije takve metode, a tu su **vremenska složenost algoritma** i **prostorna složenost algoritma**.

Vremenska složenost: Vremenska složenost algoritma kvantificira količinu vremena potrebnog algoritmu za rad kao funkciju duljine ulaza. Imajte na umu da je vrijeme izvođenja funkcija duljine unosa, a ne stvarnog vremena izvođenja stroja na kojem se algoritam izvodi. Kako bi se izračunala vremenska složenost algoritma, pretpostavlja se da je potrebno konstantno vrijeme **c** za izvršenje jedne operacije, a zatim se izračunavaju ukupne operacije za ulaznu duljinu na **N**.

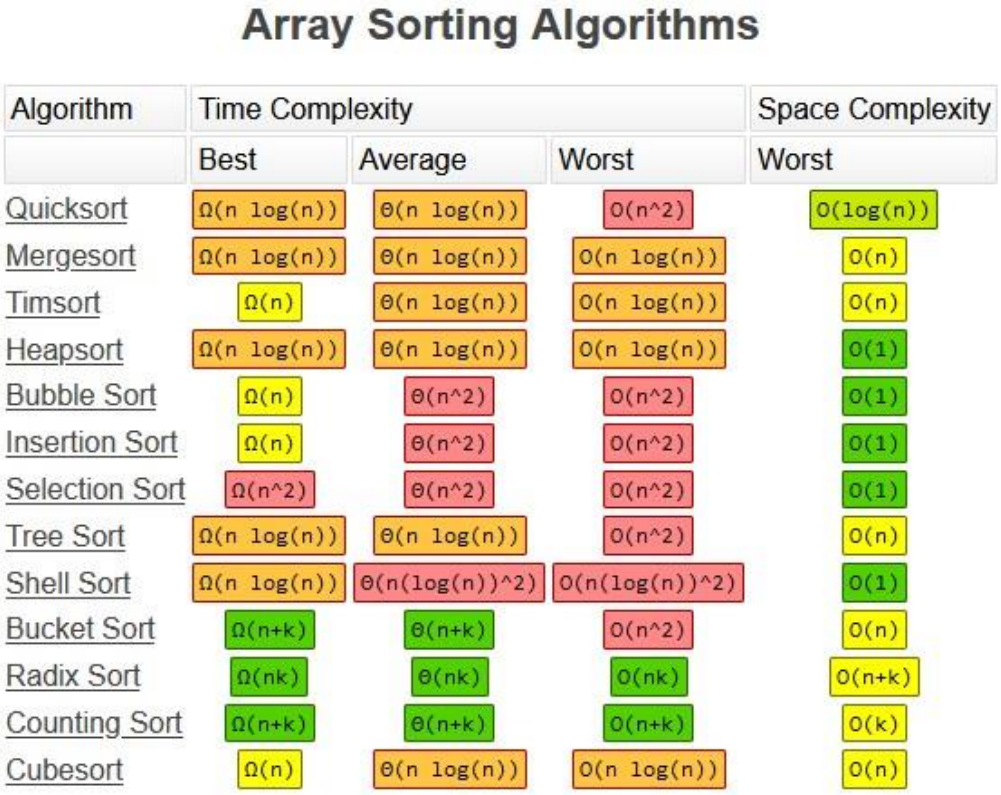
Razmotrimo primjer da bi razumjeli proces izračuna: Pretpostavimo da je problem pronaći postoji li par **(X, Y)** u nizu, **A** od **N** elemenata čiji je zbroj **Z**. Najjednostavnija ideja je razmotriti svaki par i provjeriti je li zadovoljava zadani uvjet ili ne. [17]

Uz pretpostavku da svaka od operacija u računalu traje približno konstantno vrijeme, neka bude c. Broj izvedenih redaka koda zapravo ovisi o vrijednosti Z. Tijekom analize algoritma uglavnom se razmatra najgori scenarij, tj. kada ne postoji par elemenata čiji je zbroj jednak Z. U najgorem slučaju, N*c operacija je potrebno za unos. Vanjska petlja i petlja se izvodi N puta. Za svaki i, unutarnja petlja j petlja se izvodi N puta. Dakle, ukupno vrijeme izvršenja je N*c + N*N*c + c. Sada zanemarite članove nižeg reda jer su članovi nižeg reda relativno beznačajni za veliki unos, stoga se uzima samo član najvišeg reda (bez konstante) koji je N*N u ovom slučaju. Različite oznake koriste se za opisivanje ograničavajućeg ponašanja funkcije, ali budući da je uzet najgori slučaj, za prikaz vremenske složenosti koristit će se oznaka **Big-O**. Dakle, vremenska

složenost je O(N2) za gornji algoritam. Imajte na umu da se vremenska složenost isključivo temelji na broju elemenata u nizu A, tj. ulaznoj duljini, pa ako se duljina niza poveća, vrijeme izvršenja će se također povećati.

Prostorna složenost: prostorna složenost algoritma kvantificira količinu prostora koju algoritam zauzima za izvođenje kao funkciju duljine ulaza.

Pogledajmo sada na *slici 12.* vremensku i prostornu složenost za neke algoritme koji sortiraju niz:



Slika 12 – Usporedba vremenske i prostorene složenosti algoritama, preuzeto iz [30]

Tamno zelenom su označene odlične brzine, svijelo zelenom su označene dobre, žutom bojom osrednje, dok su narančastom i crvenom loše i iznimno loše.

3.3. Algoritam sortiranja

Sortiranje je tehnika preuređivanja elemenata popisa u uzlaznom ili silaznom redoslijedu, što može biti numerički, leksikografski ili bilo koji korisnički definiran redoslijed. Sortiranje je proces kroz koji podaci su poredani uzlaznim ili silaznim redoslijedom. **Sortiranje se može klasificirati u dvije vrste:**

- Interna sortiranja: Ova metoda koristi samo primarnu memoriju tijekom procesa sortiranja. Svi podaci stavke se čuvaju u glavnoj memoriji i za ovaj proces sortiranja nije potrebna sekundarna memorija. Padam podaci koje treba sortirati mogu se istovremeno smjestiti u memoriju naziva se internim sortiranjem. Postoji ograničenje za unutarnje sortiranje; mogu obraditi samo relativno male popise zbog memorije ograničenja. Postoje 3 vrste unutarnjih sortiranja, a to su:
    - SELECTION SORT (npr. Selection, Heap)
    - INSERTION SORT (npr. Insertion, Shell)
    - EXCHANGE SORT (npr. Bubble, Quick)
- Vanjska sortiranja: Sortiranje velike količine podataka zahtijeva vanjsku ili sekundarnu memoriju. Ovaj proces koristi vanjsku memoriju kao što je HDD, za pohranjivanje podataka koji ne stanu u glavnu memoriju memorija. Dakle, primarna memorija sadrži samo podatke koji se trenutno sortiraju. Sve vanjske vrste su na temelju procesa spajanja. Različiti dijelovi podataka razvrstavaju se odvojeno i spajaju zajedno. Primjer takvog sorta je Merge sort.

U ovome radu nećemo detaljno prolaziti svaki algoritam za sortiranje, s obzirom da ih postoji mnogo, a neki se koriste relativno rijetko i za specifične primjene. Proći ćemo kroz neke vrste sortiranja, te ćemo navesti njihove prednosti i mane. U *tablici 4.* usporediti ćemo Insertion, Merge, Quick i Heap sort.

Vrsta sortiranja	Prednosti	Mane
Insertion	• Za gotovo sortirane podatke, nevjerojatno je učinkovit. • Radi na mjestu, što znači da nije potrebna pomoćna pohrana • Učinkovito za male skupove podataka.	• Manje je učinkovit na listi koja sadrži veći broj elemenata • Sortiranje umetanjem zahtijeva veliki broj pomaka elemenata
Merge	• Brži je za veće popise jer za razliku od Insertion sorta ne prolazi kroz cijeli popis nekoliko puta. • Merge sort je malo brži od Heap sorta za veće skupove	• Sporije u usporedbi s drugim algoritmima sortiranja za manje skupove podataka • Prolazi kroz cijeli proces čak i ako je popis sortiran
Quick	• Quick sort je algoritam za sortiranje na mjestu. Sortiranje na mjestu znači da ne koristi dodatni prostor za pohranu za obavljanje sortiranja.	• Nestabilan
Heap	• Učinkovit • Potrošnja memorije je manja • Konzistentan	• Nestabilan • Ogromni skupovi podataka

Tablica 4 – Prednosti i mane različitih algoritama za sortiranje

3.4. **Algoritam pretraživanja**

Pretraživanje je operacija ili tehnika koja pomaže pronaći mjesto zadanog elementa ili vrijednost na listi. Za svaku se pretragu kaže da je uspješna ili neuspješna, ovisno o tome je li element koji se traži pronađen ili ne. [18]

Neke od standardnih tehnika pretraživanja su:

- Linearno pretraživanje
- Binarno pretraživanje

Linearno pretraživanje je osnovan i vrlo jednostavan algoritam pretraživanja. U linearnom pretraživanju tražimo element ili vrijednosti u zadanom nizu prelazeći niz od početka dok željeni element ili vrijednost nije pronađen. Uspoređujemo element koji se traži sa svim elementima prisutnim u nizu i kada se element uspješno podudara, vraća indeks elementa u nizu, inače vraća -1. Linearno pretraživanje primjenjuje se na nerazvrstane ili neuređene liste, kada ima manje elemenata na listi.

Karakteristike linearnog pretraživanja:

- Koristi se za nesortirane i nesređene male popise elemenata.
- Ima vremensku složenost O(n), što znači da vrijeme linearno ovisi o broj elemenata, što nije sjajno, ali nije ni strašno.
- Ima vrlo jednostavnu implementaciju.

Binarno pretraživanje je korisno kada postoji veliki broj elemenata u nizu i kada su oni sortirani. Dakle, neophodan uvjet za rad binarnog pretraživanja jest da je lista ili niz sortirana.

Značajke binarnog pretraživanja:

- Sjajno je pretraživati velike sortirane nizove.
- Ima vremensku složenost O(log n) što je vrlo dobra vremenska složenost. Također ima jednostavnu implementaciju.

U ovome radu nećemo dublje ulaziti u temu pretraživanja, s obzirom da je fokus rada na vizualizaciji struktura podataka.

4. Vizualizacija

4.1. Što je to Vizualizacija?

Prvo pitanje koje nam se javlja u ovoj cjelini je što je to vizualizacija? Nekakva službena definicija vizualizacije bi bilo da je to proces formiranja mentalne slike. Vizualizacija je prirodan način na koji učimo jako puno stvari u životu. Kad gledamo nešto, u svojoj glavi stvorimo mentalnu sliku tog objekta, pojave ili bilo čega što vidimo i onda pamtimo informacije preko te slike. S obzirom da se u ovom radu fokusiramo na poučavanje informatike i programiranja, nećemo analizirati različite vrste vizualizacije, koja svoju ulogu ima u umjetnosti i slično, već ćemo se fokusirati na vizualizaciju podataka.

Čovjekov mozak puno lakše pamti slike nego hrpu pojmova i teksta, tako da nam je učenje pomoću vizualizacije prirodno i često jednostavnije. Recimo da želimo analizirati trend rasta cijena u zadnje dvije godine, koje su u prvom slučaju prikazane kao 2000 redova u tablici, a u drugom slučaju kao neki histogram. U prvom slučaju ćemo morati jako dugo analizirati podatke, postojati će puno veća mogućnost greške, a kod kompliciranih stvari nekada će biti nemoguće donijeti zaključak. U drugom slučaju ćemo odmah moći uočiti trend promjene cijena, biti će manja mogućnost greške, te ćemo bolje razumjeti podatke koje analiziramo. Primjena vizualizacije u pravilu pomaže svugdje gdje imamo mnogo podataka.

Uobičajeno je da se vizualizacija podataka smatra relativno modernom metodom. Zapravo, grafički prikaz kvantitativnih informacija ima duboke korijene, koji sežu u povijest najranijeg pravljenja karata i vizualnog prikaza, a kasnije u tematsku kartografiju, statistiku i statističku grafiku, s primjenama i inovacijama u mnogim područjima medicine i znanosti koja se često međusobno isprepliću.

Razvoj tehnologije poboljšao i popularizirao je široku upotrebu vizualizacije podataka danas. To uključuje tehnologije za crtanje i reprodukciju slika, napredak u matematici i statistici i novi razvoj u prikupljanju podataka, predavanju, empirijskom promatranju i bilježenju. [4]

4.2. Vizualizacija u informatici

Već dugo postoji svjesnost o računalnoj znanosti kao polju budućnosti. Računalna znanost se ispreplela sa svim aspektima svakodnevnog života, te gotovo ne postoje zanimanja u kojima nije potrebno poznavanje makar osnova informatike. Veliki dio računalne znanosti čini znanje o programiranju te je to jedno od najtraženijih zanimanja na suvremenom tržištu rada. S obzirom na to, prepoznata je potreba za uvođenjem informatike u škole od najranije dobi, a popularnost informatičkih (i srodnih) studija doživljava nagli porast.

U posljednjih 20 godina, načela se i tema vizualizacije algoritama kao nešto što bi moglo pomoći pri učenju i poučavanju rada s algoritmima tijekom programiranja. Razumjeti osnovne algoritme je osnova programiranja, ali osim za osnovne koncepte vizualiziranje algoritama može pomoći i shvaćanju složenijih pojmova u višem obrazovanju.

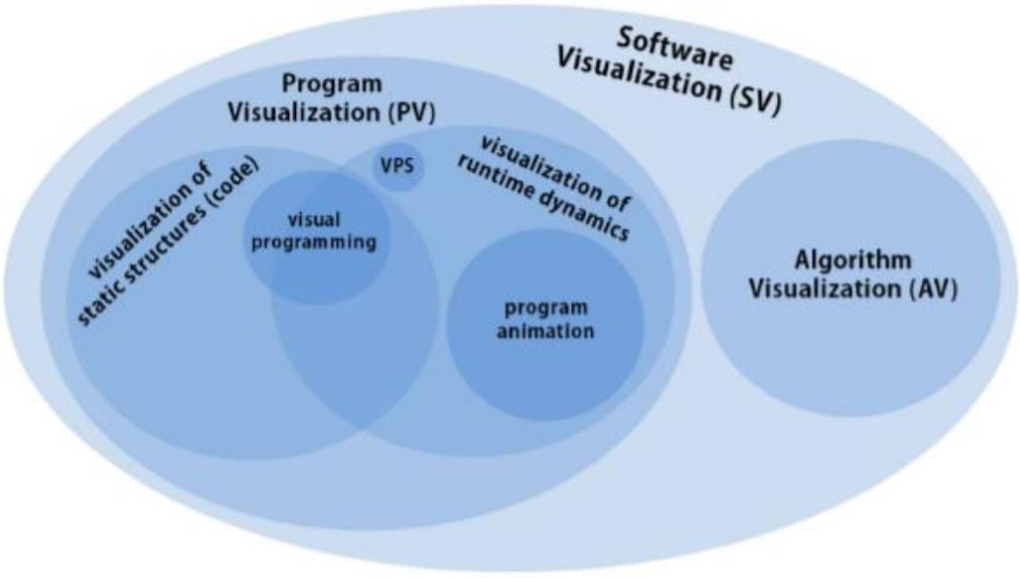
Postoje mnogi softveri koji pomažu vizualizirati osnove pojmove programiranja kao što su Scratch i mnogi softveri za vizualiziranje algoritama za sortiranje, a puno manje ljudi se bavi softverom za vizualiziranje pojmova iz podatkovne znanosti i paralelnog programiranja. To ima smisla pošto su ta polja specifična, specijalizirana i napredna, dok osnovni algoritmi su nešto kroz što će svaki programer morati proći.

Zadržavanje studenata definira uspjeh sveučilišta, veće zadržavanje dovodi do veće stope uspjeha. Kako bi se održalo visoko zadržavanje učenika u informatičkom obrazovanju, obrazovne tehnologije često se primjenjuju kako bi pomogle učenicima u učenju određene teme. Te se tehnologije obično oslanjaju na automatiziranu vizualizaciju kao svoju glavnu značajku; određena tema objašnjena je kroz intuitivnu grafiku i animaciju. [19]

Pogledajmo za početak podjelu vizualizacije softvera na dva dijela:

- Programska vizualizacija (PV)
- Algoritamska vizualizacija (AV)

Programska vizualizacija nam pokazuje izvršavanje algoritma korak po korak, dok nam algoritamska omogućuje vizualizaciju stanja struktura podataka za vrijeme izvršavanja programa.



Slika 13 – programska i algoritamska vizualizacija [20]

Na *slici 13.* vidimo podjelu raznih područja programiranja u odnosu na algoritamsku i programsku vizualizaciju.

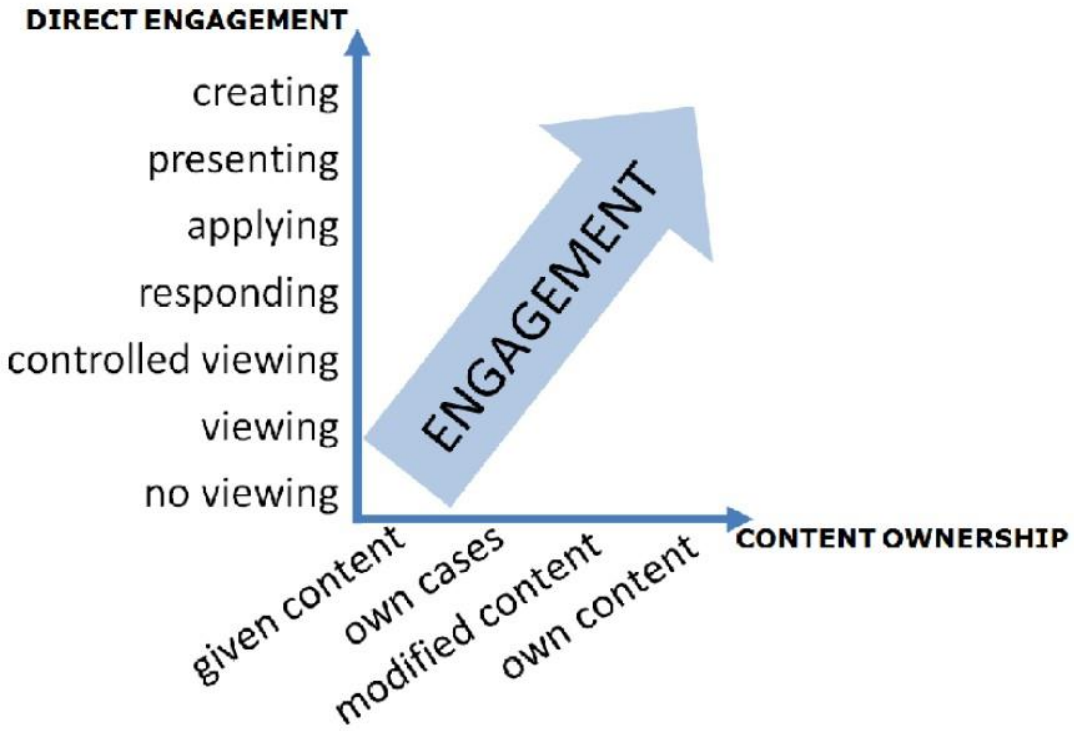
Pogledajmo sad neke karakteristike algoritamske vizualizacije:

- Pomoć nastavnicima da prikažu algoritamske operacije tijekom predavanja i konzultacija
- Pomoć studentima dok uče o osnovnim algoritmima
- Može se koristiti kao pomoć nastavnicima u pronalaženju studentskih grešaka u programima s vezanim listama
- Može se koristiti kao pomoć studentima pri učenju osnovnih operacija apstraktnog tipa podataka [21]

Proširena taksonomija učinkovitosti:

Postoji 7 razina i uglavnom se odnosi se na algoritamsku i programsku vizualizaciju.

U odnosu na vrstu sadržaja in a način i količinu angažmana, ostvariti ćemo bolji ili lošiji rezultat u radu s učenicima.



Slika 14 – proširena taksonomija vizualizacije [20]

Kao što vidimo na *slici 14*, veća razina, te bolji, orginalniji sadržaj značajno će utjecati na zainteresiranost učenika za koji slušaju naše predavanje. Vidimo da najbolji rezultat postiže kombinacija kad učenici samo izrađuju nešto, a koriste se originalni materijali, dok je najgori učinak kad nemamo prikaz već samo dobiju sadržaj.

Uloga poduke računalne znanosti

Osnovna poduka računalne znanosti pokušava pružiti detaljno razumijevanje dinamičkih procesa kao što je rad algoritma ili protok informacija između računalnih entiteta. Takvi dinamički procesi nisu dobro objašnjeni statičnim medijima kao što su tekst i slike, te ih je teško prenijeti na predavanju. Autori istražuju povijest vizualizacije u informatičkom obrazovanju, fokusirajući se na artefakte koji imaju dokumentiranu pozitivnu obrazovnu procjenu. Zatim se raspravlja o promjenama u načinu na koji je računalna tehnologija utjecala na razvoj i prihvaćanje takvih vizualizacijskih artefakata u informatičkom obrazovanju, te kako nedavne tehnološke promjene dovode do napretka u razvoju online hiperudžbenika (*eng. hypertextbooks*). [22]

4.2.1. Interaktivnost programske podrške

Programska podrška za vizualizaciju algoritama može biti napravljena na način da korisnik samo promatra rad programa, može biti napravljena da korisnik u potpunosti kontrolira program svojim unosima, i može biti sve između. Napravljen je rad [23] na tu temu koji je testirao razlike u rezultatima pre-testa i post-testa 3 grupe: grupa koja je koristila ne interaktivan alat, to jest nije bilo inputa od strane učenika, grupa koja je morala unijeti unos s vremena na vrijeme da bi program nastavio, to jest grupa s miješanom interaktivnosti, i grupa za koju je program u potpunosti ovisio o unosu korisnika. Rezultati su bili iznenađujući; nije bilo znatne promjene u rezultatima s obzirom na interaktivnost programske podrške. Daljnja analiza bi trebala biti odrađena, pošto je rezultat neočekivan i neintuitivan, tako da ovakav zaključak ne bi trebao biti olako donesen na osnovu rezultata jedne analize.

4.2.2. Nastava pojačana vizualizacijom

Nastava pojačana vizualizacijom ili VRI-moduli (*eng. Visualization-Reinforced Instruction*) predstavljaju oblik aktivnog učenja u kojem se nalazi tradicionalni nastavni materijal nadopunjen i poboljšan pažljivo dizajniranim animacijama i simulacijama ključnih koncepti, modeli i algoritmi. Kratka interaktivna vizualizacija omogućuje učenicima promatranje obrascima i odnosima koje bi inače bilo teško vidjeti.

Na primjer, parabolični ponašanje projektila lako se promatra u simulaciji projektila ili skupa projektila koji studenti se mogu poistovjetiti s igrama kao što je Angry Bird ili pucačkim igrama kao što je golf i košarka. Slično tome, redoslijed izvršavanja određenih algoritama u usporedbi s drugima (to lakše razumjeti, ali su spori ili neučinkoviti) mogu se lakše vidjeti kada se vizualiziraju kao mozaik slike koji se preslaguje nakon miješanja. Multithreading je još jedan koncept koji smo pronašli da ima koristi od VRI-ja zbog njegove apstraktne prirode.

Uloga VRI modula prilično je komplementarna i ne ometa trenutnu poduku alatima za dizajn. Moduli su pomoć za poboljšanje prostornog mapiranja gdje učenici formiraju mentalne slike pojmova koje su življi, šareniji i puni detalja zbog pojačanja koje im pruža vizualizacija. [24]

Prilikom dizajniranja VRI modula od učenika je zatraženo mišljenje o tome koje bi promjene željeli imati kako bi materijale tečaja učinili zanimljivima. Zabilježene su te promjene i uključene u dizajnirane module. Slijede karakteristike koje su studenti predložili, zajedno s vlastitom vizijom modula autora:

- Jednostavnost
- Zabava
- Kratko trajanje
- Interaktivnost
- Relevantnost
- Ponovljivost
- Informativnost

Na temelju povratnih informacija studenata, VRI moduli, kroz svoje animacije, simulacije i pokazalo se da algoritamski dizajn olakšava učenje o složenim konceptima i apstrakcijama. Početni VRI moduli bili su veliki uspjeh, ali također ističu niz problema kojima se treba pozabaviti budućnost.

S ovim dijelom ćemo završiti poglavlje vizualizacije. U sljedećem poglavlju ćemo izraditi prototip aplikacije za pomoć pri učenju struktura podataka.

5. Projekt

5.1. Cilj projekta

Cilj ovog projekta je izraditi prototip aplikacije za vizualizaciju podataka. U našem slučaju ćemo odabrati vizualizaciju binarnog stabla. Za izradu ovog projekta koristiti ćemo biblioteku (*eng. libary*) JSAV *(http://jsav.io/)* koji je napisan u Javascriptu.

Ideja u ovome radu nije izraditi potpunu aplikaciju koja će podržavati sve navedene strukture i algoritme, već na jedom primjeru pokazati mogućnosti koje nam se pružaju za izradu takve aplikacije.

JSAV podržava vizualizaciju za širok skup struktura podataka kao što su:

- Nizovi
- Vezane liste
- Stabla, binarna stabla
- Razne vrste grafova

Mana ove biblioteke je što je nastava u 2013. te nije redovito održavana, tako da dizajn relativno zastario, a nove funkcionalnosti nisu dodane. Unatoč tome, strukture podataka se nisu promijenile u zadnjih 10-tak godina, te je ova biblioteka još uvijek dobar izbor za izradu aplikacije za vizualizaciju podataka.

Biblioteka koristi:

- jQuery
- jQuery UI
- Raphael (za SVG grafiku)

Treba napomenuti da na internetu postoji još raznih biblioteka koje nam pružaju slične funkcionalnosti, a možemo očekivati da će s razvojem tehnologije njihova količina i kvaliteta rasti, te da će biti još jednostavniji i praktičniji za korištenje.

5.2. Izrada projekta

Za izradu ovog projekta prvi korak je bio preuzimanje i instaliranje JSAV biblioteke. Ono što nam je potrebno prije početka rada je:

- **Git**
- **Node.js**
- **Make** ili **Grunt** - koristi se za izgradnju JSAV biblioteke iz više izvornih datoteka.
- **uglify-js** - potreban je ako koristimo naredbu **make**

U našem slučaju odabrao sam Grunt zbog jednostavnije instalacije i bolje podrške. Za Git, Node.js i uglify-js je korištena zadnja verzija. Prvo smo klonirali repozitorij pomoću naredbe:

```
git clone https://github.com/vkaravir/JSAV.git
```

Zatim smo s korištenjem naredbe *make* napravili jedan file iz više izvornih datoteka.

S tim koracima biblioteka je spremna za korištenje, sve što sada trebamo napraviti je dodati datoteku *JSAV.js* koju smo generirali u prethodnom koraku.

Za naš projekt trebamo dodati još neke biblioteke:

- *raphael.js*
- *2.1.4/jquery.js*
- *1.11.4/jquery-ui.min.js*

```
44    <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.js"></script>
45    <script src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.4/jquery-ui.min.js"></script>
46    <script src="../lib/jquery.transit.js"></script>
47    <script src="../lib/raphael.js"></script>
48    <script src="../build/JSAV.js"></script>
```

Slika 15 – potrebne datoteke za izradu project

Treba napomenuti, da cilj ovog rada nije izraditi potpuni program od nule, već na kratkom demo primjeru pokazati dio mogućnosti koju nam pruža ova biblioteka, pri

izradi alata za pomoći pri vizualizaciji. Također ćemo na kraju ovog rada spomenuti alternative korištenju JSAV biblioteke.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example of Binary Tree</title>
    <link rel="stylesheet" href="../css/JSAV.css" type="text/css" media="screen" title="no title" charset="utf-8" />
    <style>
    #av {…
    }
    .jsavcounter {…
    }
    .jsavtree {…
    }
    svg {…
    }
    path {…
    }
    </style>
  </head>
  <body>
    <h1>JSAV slideshow for binary trees</h1>
    <div id="av">
      <div class="jsavcontrols"></div><span class="jsavcounter"></span>
    </div>
```

Slika 16 – head i div elementi koda

Na *slici 16.* vidimo da je HTLM i CSS dio minimalan, imamo samo neke osnovne stilove, jedan "container" s oznakom "av" u kojemu ćemo crtati naše stablo, te jedan dio u kojemu su naredbe za listanje koraka.

U ovome demo primjeru napraviti ćemo binarno stablo za koje ćemo zadati fiksne vrijednosti, te pokazati kako ih dodajemo korak po korak. Cilj takvog demo primjera je učenicima kroz demonstaciju pokazati odnos roditelj-dijete kod čvorova u stablu, pokazati kako binarno stablo može imati najviše dva djeteta, te kad završimo crtanje stabla postaviti im pitanja da provjerimo usvojeno znanje.

Na *slici 17.* viditi ćemo kod izrade binarnog stabla korištenjem JSAV biblioteke.

Prvo trebamo inicijalizirati "jsav" varijablu, a zatim ćemo napraviti binarno stablo, gdje ćemo ga inicijalizirati pozivom funkcije `.binarytree()`.

S obzirom da u ovome primjeru koristimo dijaprojekciju, nakon svakog koraka dodajemo `jsav.step()`, pomoću kojega jsav "pamti" taj korak i kasnije omogućuje listanje. `bt.layout()` će nam osvježiti sliku.

Na početku ćemo inicijalizirati korijen, pomoću funkcije koju vidimo na liniji 42. Na liniji 46 smo korijen obojali žutom bojom, ta opcija može biti korisna kad želimo implementirati pretragu ili neke zadatke pomoću ove biblioteke.

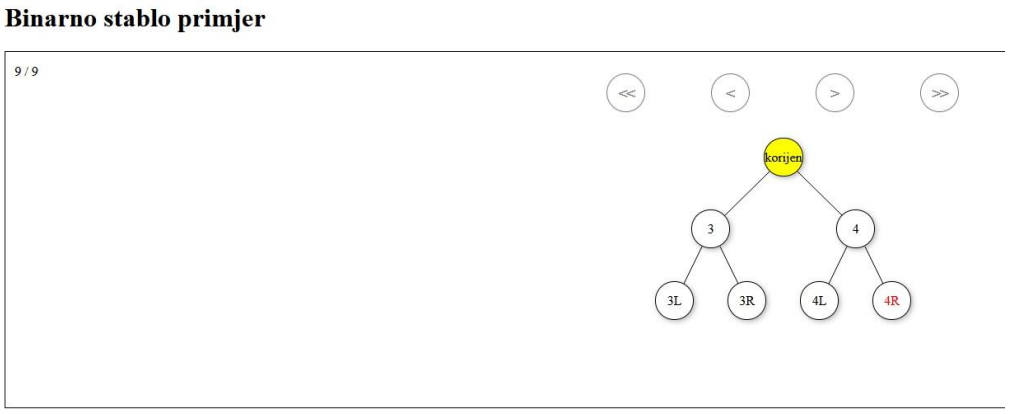
Na liniji 47 vidimo kako je jednostavno dodati novo dijete, samo odaberemo da li je lijevo ili desno, te kao props proslijedimo njegovu vrijednost. S obzirom da se radi o binarnom stablu, ako pokušamo dodati ponovo lijevo ili desno dijete, samo ćemo pregaziti prethodni podatak.

```
39  var jsav = new JSAV("av");
40  var bt = jsav.ds.binarytree();
41  jsav.step();
42  bt.root("korijen"); // postavi vrijednost korijena
43  bt.layout(); // update layout
44  jsav.step(); // dodaj novi korak
45
46  bt.root().css({"background-color": "yellow"}); //postavi boju korijena na žutu
47  bt.root().left("3"); // postavi lijevo dijete na vrijednost 3
48  bt.layout(); //..update layout
49  jsav.step();
50
51  var right = bt.root().right("4"); // postoavi desno dijete, spremi referencu
52  bt.layout();
53  jsav.step();
54
55  right.left("4L"); // postavi lijevo dijete od prethodnog čvora
56  bt.layout();
57  jsav.step();
58
59  bt.root().left().left("3L").parent().right("3R"); // ulančavanje, postavi lijevo i desno dijete
60  bt.layout();
61  jsav.step();
62
63  var node = bt.newNode("4R");
64  node.css({"color": "red", "top": right.css("top"), "left": parseInt(right.css("left"), 10) + 60});
65  jsav.step();
66
67  right.right(node);
68  bt.layout();
69
70  jsav.recorded(); // spremi sve promjene i omogući vraćanje
```

Slika 17 – kod izrade stabla pomoću JSAV biblioteke

Na liniji 49 smo pokazali primjer kako možemo ulančati više naredba za postaviti dva čvora u jednoj liniji, te smo također koristili svojstvo roditelja kako bi dohvatili čvor.

Na kraju smo spremili promjene kako bi mogli listati po koracima.



Slika 18 – izgled nacrtanog binarnog stabla iz primjera

5.3. Drugi alati za vizualizaciju

Sada ćemo spomenuti još neke opcije koje nam pruža ova biblioteka, te ćemo spomenuti neke druge biblioteke i alate koji nam pružaju slične funkcionalnosti.

Kod ove biblioteke imamo sljedeće strukture na raspolaganju: API liste, strukture podataka, čvorovi i rubovi, API grafa, API povezane liste, API matrice, API stabla i binarnog stabla

VisuAlgo

Od ostalih alata, vjerojatno najbolji gotov alat koji je trenutno na raspolaganju je **VisuAlgo** (https://visualgo.net/en).

Ovaj alat su razvili na nacionalnom sveučilištu u Singapuru, 2011. godine pod vodstvom dr. Stevena Halima. U trenutku pisanja ovog rada, još uvijek ne podržava potpuno rad na mobilnim uređajima, ali je to nešto što je u procesu i u budućnosti možemo očekivati. Stranica se redovito održava i dodaju se novi sadržaji, te je od svih alata za vizualizaciju podataka u ovom trenutku najbolja.

Nudi razne vrste struktura i algoritama, gdje možemo interaktivno dodavati, brisati, sortirati ili pretraživati određenu strukturu, ići korak po korak, kontrolirati brzinu i još mnogo toga. U budućnosti je planirano i dodavanje kviza kako bi se provjerilo usvojeno znanje učenika. Ono što je posebno privlačno kod ovog alata je moderan UI i dizajn, što nije slučaj u većini drugih stranica ili biblioteka, kao a primjer i u našem projektu.

Data Structure Visualizations

Sljedeći alat vrijedan spomena je "Data Structure Visualizations"(https://cmps-people.ok.ubc.ca/ylucet/DS/Algorithms.html) koji je napravljen na sveučilištu u San Franciscu. Ova stranica podržava preko 50 kompleksnih programa koje možemo vizualizirati i naučiti.

Prednost ove stranice je da nam pruža vizualizaciju dosta kompleksnijih problema u programiranju, dok se VisuAlgo preporučuje kod učenja osnovnih stvari. Možemo reći da je VisuAlgo bolji za početnike, dok je Data Structure Visualizations odličan izbor za već iskusnije polaznike, koji žele naučiti rekurziju, radix stabla i slično.

The Pathfinding Visualizer

Ovaj Github projekt koji je napravio Clément Mihailescu posvećen je grafovima. Na velikoj "mapi" podijeljenoj na kockice, označimo zidove tj. prepreke, te početak i kraj.

U svojoj srži, algoritam za traženje puta nastoji pronaći najkraći put između dvije točke. Ova aplikacija vizualizira različite algoritme za pronalaženje puta.

Jako je zanimljiva učenicima, jer samo prave svoj labirint i onda gledaju kako ga različiti algoritmi rješavaju.

Toptal

Zadnju stranicu koju ćemo spomenuti u našem radu je toptal (https://www.toptal.com/developers/sorting-algorithms). Ova stranica je odlična za vizualizaciju algoritama sortiranja, te usporedbu brzine za različite podatke, možemo vidjeti učinkovitost s obzirom da li su podaci već skoro sortirani i slično.

S ovim dijelom završavamo naš rad, svaki od navedenih programa i alata ima svoje prednosti i mane, Toptal je odličan za usporedbu algoritama sortiranja i može biti koristan i iskusnijem programeru, VisuAlgo je odličan ja početnike, The Pathfinder Visualizer uči algoritme praktički kroz "igru", a Data Structure Visualizations pruža najširi izbor algoritama. Ako pak želimo veću slobodu kod izrade vlastitih materijala, možemo se odlučiti za JSAV biblioteku ili neku sličnu opciju.

6. Zaključak

U ovome radu upoznali smo se s osnovnim strukturama podataka, te algoritmima za njihovo sortiranje i pretraživanje. U prvom poglavlju smo objasnili šta je to niz, vezana lista, stog, red, stablo, binarno stablo i graf, te naveli njihove prednost, mane i primjene. Zatim smo u sljedećem poglavlju obradili neke algoritme za sortiranje i pretraživanje tih struktura podataka. U trećem poglavlju dolazimo do vizualizacije, koja je odlična metoda za učenje raznih tema u programiranju, a posebno zahtjevnih tema kao ova. U zadnjem poglavlju pravimo prototip aplikacije za vizualizaciju struktura podataka, za čiji smo primjer uzeli binarno stablo.

Ovaj projekt je samo prototip, a u budućnosti ga je cilj izmijeniti i nadograditi na način da podržava sve strukture podataka koje smo naveli u ovome radu, redizajnirati UI i dodati nove opcije, kao na primjer testove za studente koji će moći provjeriti svoje znanje. Također je cilj projekt prebaciti iz običnog Javascripta u React, kako bi njegovo razvijanje i održavanje bilo lakše.

Ono što možemo zaključiti je da je vizualizacija od izrazite pomoći pri učenju ove teme, čak bi se moglo reći neophodna za potpuno razumijevanje. Svi oni koji poučavaju strukture podataka i algoritama trebali bi koristi neki oblik vizualizacije za ovu temu, kako bi olakšali proces učenja.

U ovome radu naveli smo neke stranice i alate koje možemo koristiti za vizualizaciju, kao što su VisuAlgo, Data Structure Visualizator, The Pathfinder Visuaizer i Toptal. Svaki od njih ima svoje prednosti i svoju primjenu.

S razvojem tehnologije, a posebno umjetne inteligencije, možemo očekivati da će se u budućnosti pojaviti još više naprednih alata koji će dodatno olakšati proces učenja.

7. Reference

[1] geeksforgeeks.org, 2022. [Mrežno]. Available: https://www.geeksforgeeks.org/data-structures/.

[2] J. K., »commonsense.org,« 2022. [Mrežno]. Available: https://www.commonsense.org/education/articles/teachers-essential-guide-to-coding-in-the-classroom.

[3] D. Loshin, 2022. [Mrežno]. Available: https://www.techtarget.com/searchdatamanagement/definition/data-structure.

[4] M. Friendly, A Brief History of Data Visualization, 2006.

[5] www.studytonight.com, 2022. [Mrežno].

[6] »https://www.geeksforgeeks.org/applications-of-linked-list-data-structure/,« 2022. [Mrežno].

[7] »https://www.geeksforgeeks.org/queue-linked-list-implementation/,« 2022. [Mrežno].

[8] S. Pandey, "Data Structures in a Nutshell", Rochester, NY., 2020.

[9] S. Mohapatra, DATA STRUCTURES USING C, Biju Patnaik University of Technology, Odisha.

[10] S. S. Skiena, The Algorithm Design Manual, Springer Science & Business Media, 2009.

[11] baeldung, 2021. [Mrežno]. Available: https://www.baeldung.com/cs/applications-of-binary-trees.

[12] M. F. Khawaja, 2021. [Mrežno]. Available: https://www.makeuseof.com/what-are-binary-trees-and-why-should-you-know-them/.

[13] 2022. [Mrežno]. Available: https://www.geeksforgeeks.org/introduction-to-graphs-data-structure-and-algorithm-tutorials/.

[14] J. Bullinaria, Data Structures and Algorithms, Birmingham, 2019.

[15] m. university. [Mrežno]. Available: https://mu.ac.in/wp-content/uploads/2021/05/Data-Structure-Final-.pdf.

[16] 2022. [Mrežno]. Available: https://www.geeksforgeeks.org/time-complexity-and-space-complexity/.

[17] [Mrežno]. Available: https://www.srividyaengg.ac.in/coursematerial/ECE/106325.pdf.

[18] M. Ayub, R. A. Nathasya i O. Karnalim, »Integrating program and algorithm visualisation for learning data structure implementation,« 2019.

[19] Sorva, Karavirta i Malmi, A review of generic program visualization systems for introductory programming education. ACM Transactions on Co mpu ting, 2013.

[20] M. Mladenović, »skripta - vizualizacija u nastavi infromatike,« 2022.

[21] E. Fouh, M. Akbar i C. A. Shaffer, »The Role of Visualization in Computer Science Education,« 2012.

[22] P. R. O. Z. K. i E. O. , »Algorithm Visualization Environments: Degree of interactivity as an influence on student-learning?, 2019.

[23] Dr. Mahmoud K Quweider, »Visualization as Effective Instructional and Learning Tools in the Computer«.

[24] »https://www.studytonight.com/data-structures/linked-list-vs-array,« [Mrežno].

[25] »https://medium.com/analytics-vidhya/queue-deque-overview-and-its-implementation-in-python-c36c56b532b8,« [Mrežno].

[26] »https://www.gpp7.org.in/wp-content/uploads/sites/22/2020/04/file_5ea6ee3012f66.pdf,« [Mrežno].

[27] »https://forum.huawei.com/enterprise/en/machine-learning-algorithms-decision-trees/thread/710283-895,« [Mrežno].

[28] »https://media.geeksforgeeks.org/wp-content/uploads/20200630130949/applications_graph.jpg,« [Mrežno].

[29] C. P. W. j. K. i H. S. L. , »An Efficient Approach Using Knowledge Distillation Methods to Stabilize Performance in a Lightweight Top-Down Posture Estimation Network,« *Cosimo Distante,* 2021.

[30] »https://lamfo-unb.github.io/2019/04/21/Sorting-algorithms/,« [Mrežno].