Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/06BE48Radisic

IMPLEMENTACIJA VIZUALIZATORA STRUKTURA PODATAKA KORIŠĆENJEM WEBASSEMBLY PLATFORME

DATA STRUCTURE VISUALIZATION IMPLEMENTATION USING WEBASSEMBLY PLATFORM

Rade Radišić, Srđan Popov, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Izučavanje struktura podataka može da bude apstraktno za studente koji su početnici u programiranju. Njihovo razumevanje memorijskog modela računara nije na dovoljno visokom nivou. Strukture podataka temelje se na teoriji grafova, koji se na lak način mogu vizualizovati i intuitivno razumeti. Tema ovog rada je softverska implementacija vizualizatora za strukture podataka.*

Ključne reči: *Vizualizacija struktura podataka, Rust, JavaScript, WebAssembly*

Abstract – *Studying of data structures can be abstract for beginner student programmers. Their understanding how computer memory works is vague. Data structures are based on graph theory, which could be easily visualized and understood intuitively. The subject of this paper is a software implementation of a data structure visualization.*

Keywords: *Data structure visualization, Rust, JavaScript, WebAssembly*

1. UVOD

Strukture podataka su, zajedno sa algoritmima, fundamentalni principi računarskih nauka. Budući softverski inženjeri bave se njihovim izučavanjem na početku svog školovanja. Da bi se efikasno izučavali algoritmi i strukture podataka, neophodna su određena znanja iz diskretne matematike kao što su poznavanje formalne logike, skupovi, relacije, funkcije i teorija grafova [1]. Matematičke definicije i zapisi pretenduju da budu apstraktni za nekog kome je bitno da se sa pragmatičnog stanovišta upozna sa upotrebom algoritama i struktura podataka. Taj problem moguće je izbeći korišćenjem vizualizacija, kao pomoćnim sredstvom. Graf G je uređen par $(V(G), E(G))$ gde su $V(G)$ skup njegovih čvorova i $E(G)$, disjunktan od $V(G)$, skup njegovih ivica zajedno sa funkcijom učestalosti ψ_G, koja formira svaku ivicu grafa G kao (ne)uređen par čvorova grafa G [2].

U zavisnosti od tipa relacije, grafovi mogu biti usmereni i neusmereni. Jednostavnije objašnjenje grafova bi bilo da su čvorovi tačke, a svaka ivica je prikazana kao linija koja spaja dva čvora. Vizualni prikaz grafova može biti upotrebljen kao sredstvo za vizualizaciju struktura podataka, jer se one definišu kao usmereni grafovi.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Srđan Popov, vanr. prof.

Prikazivanje podataka je bitna problematika kojim se računarstvo bavi. U tu svrhu razvijana su mnogobrojna softverska rešenja. Njihova rasprostranjenost i upotrebljenost zavisila je od faktora kao što su multiplatformska podrška i otvorenost koda.

Moderno računarstvo fokusira se na izradu web aplikacija, jer se one koriste i prikazuju u internet pretraživačima, čiji je osnovni nefunkcionalni zahtev da budu multiplatformske, odnosno, da se izvršavaju na identičan način na različitim operativnim sistemima i uređajima.

Postoji više pristupa koji se primenjuju prilikom izučavanja programiranja i programskih jezika. Pristup „odozdo prema dole" bio bi da student uči programiranje u programskom jeziku visokog nivoa, kao što su *Python*, *Java* ili *Javascript*. Ove i njima slične jezike odlikuje osobina postojanja velikog broja apstrakcija, koje uprošćavaju sam jezik i time ga čine lakšim za učenje i dalje korišćenje. Cena tih apstrakcija je da su autori jezika napravili izbor kako se rukuje primitivnim operacijama. Primer ovakve apstrakcije je postojanje mehanizma automatskog rukovanja memorijom, *Garbage Collection*. Pristup "odozdo prema gore" podrazumeva učenje jezika niskog nivoa[1] poput programskog jezika *C*. Prilikom rada sa ovakvim jezicima, programer mora da vodi računa o dinamički alociranoj memoriji, odnosno da je oslobodi kada ona više nije neophodna.

Prethodno opisani pristupi direktno utiču na studentovo razumevanje struktura podataka. U prvom slučaju, on će imati osnovno razumevanje rada sa strukturama podataka, dok će u drugom dobiti i tačnu sliku stanja u memoriji prilikom rada. Problem kod drugog slučaja nalazi se u tome što je potrebno više vremena pre nego što student ovlada principima koji stoje iza struktura podataka. Trenutno stanje je da nijedan od dva navedena pristupa nije dovoljno dobar, te su, u svrhu smanjenja jaza između pristupa učenju, autori razvili softverski alat pomoću kog se mogu vizualizovati strukture podataka pomoću vizualizacije usmerenih grafova.

2. PREGLED KORIŠĆENIH TEHNOLOGIJA

2.1. WebAssembly

WebAssembly je mašinski format instrukcija za virtuelnu mašinu baziranu na stek arhitekturi [3]. Ona omogućava drugim programskim jezicima koji nisu *JavaScript* da se

[1] Pod jezicima niskog nivoa uglavnom se misli na asembler i mašinski jezik računara. Pod ovom konotacijom, podrazumevaju se jezici u kojim se, pre svega, manuelno upravlja memorijom koju koriste, ali su za razliku od asemblerske grupe jezika, platformski nezavisni.

izvršavaju u okviru internet pretraživača [4]. *WebAssembly* podržavaju svi najpopularniji internet pretraživači, ali implementacija same virtuelne mašine postoji i van njih, te ju je moguće koristiti i za samostalne aplikacije. Koncept virtuelne mašine je veoma sličan bajtkodu *Java* virtuelne mašine (*JVM*), kao i *IL Assembler*-u *.NET Framework*-a [3].

Prednosti ovakve implementacije u odnosu na *JavaScript* sastoji se u tome što se, pre svega, radi o mašinskom formatu, dok je *JavaScript* kod tekstualan i interpretiran. To uključuje dodatne korake prilikom izvršavanja *JavaScript* koji uključuju lekser i parser jezika, dok je binarni format *WebAssembly*-ja odmah moguće izvršiti od strane virtuelne mašine.

WebAssembly koristi linearni model memorije. U pitanju je kontinualni blok bajtova, koji može biti deklarisan unutar *WebAssembly* modula, izveden iz modula ili uvezen iz platforme na kojoj se *WebAssembly* izvršava [3]. Na ovaj način *WebAssembly* i *JavaScript* međusobno komuniciraju. Važno je napomenuti da *WebAssembly* još uvek ne može direktno da manipuliše *Document Object Model* (*DOM*) stablom, te se kroz linearnu memoriju mogu proslediti podaci pomoću kojih će *JavaScript* kod ažurirati *DOM* stablo.

Virtuelna mašina *WebAssembly*-ja je veoma jednostavna, tako da ne podržava *heap* memoriju, odnosno nema dinamičku alokaciju objekata pomoću *new* operatora. Umesto toga, postoje četiri primitivna tipa koja se koriste [3]:

Tabela 1. *Tipovi podataka u WebAssembly-ju*

i32	32-Bit Integer
i64	64-Bit Integer
f32	32-Bit Floating-Point Number
f64	64-Bit Floating-Point Number

WebAssembly poseduje tekstualnu sintaksu koju je moguće pisati i prevoditi u binarni format. Ona umnogome podseća na *LISP* programski jezik. Uglavnom nije slučaj da programer direktno piše takav kod, već se jezici visokog nivoa prevode, a ciljani jezik je binarni format koji virtuelna mašina može da izvršava. Trenutna situacija je da jezici čije je upravljanje memorijom automatsko putem korišćenja mehanizma *Garbage Collection* nisu podržani, ili nisu podržani u potpunosti. Neki od takvih jezika koji su podržani su programski jezik *Go*, *C#* i *AssemblyScript*, koji je jezik dijalekat *TypeScript*-a. Nasuprot jezicima sa automatskim upravljanjem memorijom, jezici koji nemaju *Garbage Collection* kao što su programski jezici *C*/*C++* i *Rust* su podržani i najčešće se koriste priliko rada sa *WebAssembly*-jem.

2.2. Rust programski jezik

Rust [4] programski jezik je moderan programski jezik. Pojavio se kao sistemski jezik, ali je ubrzo našao primenu i u drugim oblastima kao što je razvoj web aplikacija. Glavna osobina po čemu se razlikuje od drugih jezika je tzv. *ownership* sistem, koji predstavlja način manuelnog upravljanja memorijom, ali na način da su pravila implementirana u samom kompajleru, odnosno, ukoliko je neko od pravila za upravljanje memorijom prekršeno, program se neće kompajlirati.

Ownership sistem je alternativa automatskim upravljanjem memorijom kod drugih modernih programskih jezika, koji omogućava visoke performanse programa, dok sa druge strane, model upravljanja memorijom je urađen tako da uslovljava programera da piše memorijski bezbedan kod. Mana *ownership* sistema je što čini da kriva učenja/usvajanja jezika bude strmija, odnosno, potrebno je više vremena kako bi se shvatili razlozi principa koji iza njega stoje.

Podrška za rad sa *WebAssembly*-jem je raznovrsna za programski jezik *Rust*. *wasm-bindgen* je biblioteka i komandni alat u programskom jeziku *Rust* koja služi za interakciju između *WebAssembly* modula i *JavaScript*-a [5]. *web-sys* i *js-sys* su biblioteke pomoću kojih se iz *Rust* koda mogu pozivati funkcije u *JavaScript*-u za manipulaciju *DOM* stablom i funkcije standardne *Javascript* biblioteke, respektivno.

2.3. JavaScript biblioteka D3.js

Za vizualizaciju grafova korišćena je *D3.js JavaScript* biblioteka, kao jedno od najboljih i dokazanih rešenja za vizualizaciju podataka u *JavaScript*-u.

D3.js podržava prikazivanje podataka iz *JavaScript Object Notation* (*JSON*) objekata, kao i manipulaciju vektorskim formatom, odnosno *Scalable Vector Graphics* (*SVG*). Velika pogodnost ovog vektorskog formata je što je podržan u svim internet pretraživačima.
D3.js funkcioniše po principu vezivanja podataka za *DOM* stablo i primenom podacima vođenim transformacijama nad njime [6]. Nije monolitan radni okvir koji pokušava da odgovori svakom zahtevu, već je fleksibilno rešenje, koje daje osnovu za manipulaciju *DOM* stablom, a na onome ko je koristi je da upotrebi različite web standarde zarad uspešne vizualizacije.

3. DATA STRUCTURE VISUALIZER TOOL

3.1. Pregled osnovnih funkcionalnosti

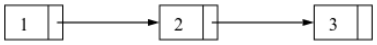
Alat je razvijen u svrhu lakšeg savladavanja rada sa strukturama podataka, na osnovu problematike objašnjene u uvodnom delu. Sastoji se od dva interfejsa:

- "Interfejs studenta", grafički,interaktivni interfejs
- "Interfejs nastavnika", tekstualni jezik specifičan za domen (*Domain Specific Language - DSL*), koji se koristi za definiciju strukture podataka

Interfejs studenta je interaktivna, grafička aplikacija u kojoj je moguće manipulisati strukturama podataka. Student može da izabere rad sa više različitih struktura podataka.

Deo za rad sa konkretnom strukturom podataka sastoji se iz dva dela: grafičkog prikaza trenutnog stanja strukture podataka i panela koji prikazuje moguće operacije.
Na primeru spregnute realizacije steka sa slike, vidi se da postoje operacije *push* i *pop*.

Unosom konkretne vrednosti i pritiskom "Push" dugmeta, dodaće se vrednost na kraj steka, dok pritiskom na dugme "Pop" biće uklonjena vrednost sa kraja steka.



Slika 1. *Primer spregnute realizacije steka*

Interfejs nastavnika koristi se za definiciju struktura podataka koje će student koristiti. Nastavnik ima na raspolaganju sve neophodne operacije, a na njemu je da izabere koje operacije će i gde biti korišćene. Kao dodatni nivo fleksibilnosti, nastavnik može posebno nazvati izabrane operacije za datu strukturu podataka, kao i samu strukturu podataka.

Ova opcija dodata je iz razloga što različite literature iste strukture podataka i njihove operacije nazivaju različitim imenima, tako da se alat može prilagoditi terminologiji koja se koristi u nastavi. Konkretan primer koda jezika specifičnog za domen koji bi nastavnik koristio za definiciju strukture podataka je sledeći:

```
datastructure!                          {
      name                          "Stack"
      type                         "linear"
      implementation               "linked"
      operation    "Push",    push_back
      operation     "Pop",     pop_back
      operation     "Clear",      clear
 }
```

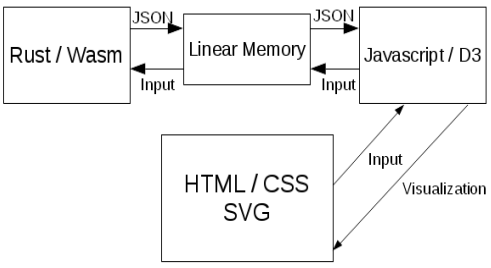
Primer 1. *Primer definicije spregnute realizacije steka*

Alat je upotrebljiv iz internet pretraživača, bez ikakve dodatne instalacije.

3.2. Arhitektura softverskog rešenja

Graf je modelovan pomoću struktura u programskom jeziku *Rust* tako da se sastoji od dva vektora: jedan za čvorove, drugi za ivice grafa. Svaka operacija nad grafom biće sračunata na *Rust* strani, odnosno, unutar *WebAssembly* virtualne mašine.

Nakon dobijanja novog stanja grafa, ono će biti konvertovano u *JSON*, što će dalje biti preneseno kroz linearnu memoriju do *JavaScript* koda, gde će *D3.js* funkcije biti zadužene za iscrtavanje grafa na osnovu dobijenih podataka. Funkcije za definisanje interfejsa su takođe na *Rust* strani, ali aktivno koriste funkcije iz *web-sys* biblioteke, kako bi se manipulisalo *DOM* stablom.



Slika 2. *Arhitektura vizualizatora struktura podataka*

Za komunikaciju između panela sa komandama i koda napisanog u *Rust*-u koristi se *requestAnimationFrame* metoda, čija je svrha da bude *event loop* i čeka događaje koji se generišu pritiskom na dugmad. Onog momenta kada se pojavi odgovarajući događaj, on će biti pročitan od strane *event loop-a*, na osnovu čega će biti izvršena odgovarajuća akcija, koja će rezultirati promenom stanja grafa, gde će se generisati novi *JSON* objekat, koji će biti poslat na iscrtavanje. Tako će biti promenjen prikaz grafa. Kako je prethodno pomenuto, tekstualni jezik specifičan za domen koristi se kako bi se definisala konkretna struktura podataka.

Primer 1. daje izgled jedne takve definicije. U pitanju je implementacija pomoću *Rust* makroa, gde će se makro transformisati u kod koji koristi *Builder* softverski obrazac [7]. Definicija strukture iz primera 1. prevešće se u sledeći kod u programskom jeziku *Rust*:

```
LinearStructureBuilder::new(
    "Stack".to_string()
).push_back("Push".to_string()).unwrap())
 .pop_back("Pop".to_string()).unwrap()
 .build();
```

Primer 2. *Prevođenje koda definicije strukture podataka iz* Rust *makroa*

Rezultat *wasm_bindgen* build-a je *WebAssembly* modul zapakovan kao *Node Package Manager* (*NPM*) paket. U tom obliku, moguće ga je uključiti u bilo koju modernu *JavaScript* frontend aplikaciju koja koristi *NPM* za upravljanje zavisnostima.

4. ZAKLJUČAK

Vizualizacija je moćno sredstvo koje pomaže pri usvajanju novih koncepata. Strukture podataka su usmereni grafovi, te se njihove vizualizacije mogu koristiti kao sredstvo za objašnjavanje struktura podataka, koje su, inače vrlo apstraktan pojam. Alat koji je predstavljen je još u ranom razvoju.

Dalji planovi razvoja sastoje se u implementaciji svih struktura podataka, kao i omogućavanje većeg nivoa interaktivnosti, što bi dalje rezultovalo podrškom za vizualizaciju algoritama, odnosno, njihovo izvršavanje „korak po korak”. Dobar primer kakvom alatu autori teže razvijen je na univerzitetu u San Francisku. To vizualno rešenje razvija se već dugi niz godina i promenilo je i nekoliko tehnologija, gde je trenutno aktuelno rešenje u *JavaScript*-u.

Ono što je mana tog rešenje jeste nemogućnost prilagođavanja terminologije, za razliku od rešenja prikazanog u ovom radu.

Mogućnosti koje pruža *WebAssembly* su velike, jer omogućava ostalim jezicima opšte namene da se izvršavaju unutar internet pretraživača. Apetiti tima koji radi na *wasm* platformi su veći, te ga oni vide kao ciljnu platformu za izvršavanje softvera odgovornog za rad *cloud* sistema. Sa druge strane, mnoge biblioteke i softveri su na ovaj način našle put do izvršavanja u internet pretraživačima, samim tim je težnja da se kod piše za jednu, uniformnu platformu dostigla novi nivo.

5. LITERATURA

[1] Prof. Tom Leighton, Dr. Marten van Dijk "Mathematics for Computer Science", MIT course

[2] J.A. Bondy, U.S.R. Murty "Graph Theory", 3rd edition, 2008

[3] Kevin Hoffman "Programming WebAssembly with Rust"

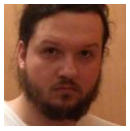
[4] Steve Klabnik, Carol Nichols "The Rust Programming Language"

[5] "wasm-bindgen guide", official wasm-bindgen documentation

[6] "D3.js - Data-Driven Documents", official D3.js documentation

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides "Design Patterns Elements of Reusable Object-Oriented Software"

Kratka biografija:



Rade Radišić rođen je u Zrenjaninu 1992. godine, gde je završio osnovnu i srednju elektrotehničku školu. 2011. godine upisuje Fakultet tehničkih nauka, smer Računarstvo i automatika. 2015. godine brani diplomski rad i upisuje master studije. Položio je sve ispite predviđene planom i programom.