Programski jezik julia

Stojanović, Josip

Sveučilište u Rijeci, Fakultet informatike i digitalnih tehnologija

Sveučilišni prijediplomski studij Informatika

Josip Stojanović

Programski jezik Julia

Završni rad

Mentor: Prof. dr. sc. Ana Meštrović

Rijeka, 25.9.2023.

Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija
www.inf.uniri.hr

Rijeka, 21.6.2023.

Zadatak za završni rad

Pristupnik: Josip Stojanović

Naziv završnog rada: Programski jezik Julia

Naziv završnog rada na engleskom jeziku: Programming language Julia

Sadržaj zadatka:

Zadatak završnog rada jest dati kratki pregled programskog jezika Julia. Potrebno je opisati osnovne značajke jezika i istaknuti na koji način jezik Julia podržava funkcijski stil programiranja. U drugom dijelu rada potrebno je prikazati primere primjene jezika Julia kroz rješavanje nekoliko odbaranih problemskih zadataka.

Mentor

Prof. dr. sc. Ana Meštrović

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 22.6.2023.

(potpis pristupnika)

Adresa: Radmile Matejčić 2
51000 Rijeka, Hrvatska
Tel: **+385(0)51 584 700**
E-mail: **ured@inf.uniri.hr**
OIB: **64218323816**
IBAN: **HR1524020061400006966**

Sadržaj

Uvod....4
Pregled programskog jezika Julia....5
Povijest i razvoj jezika julia....5
Početci i motivacija....5
Dizajn jezika....5
Open-source filozofija....5
Osnovne značajke jezika julia....6
Brzina izvršavanja i JIT kompajliranje....6
Multiple dispatch....6
Dinamički tipovi s opcionalnim anotacijama tipova....6
Paralelno računanje i distribuirano računarstvo....6
Analiza jezika Julia....7
Kontrolne strukture....7
Tipovi podataka u juliji....8
Vrste tipova podataka....8
Stablo tipova:....8
Prednosti stabla tipova u juliji:....9
Osnovni tipovi podataka....9
Kolekcije:....10
Složeni tipovi podataka:....10
Rad sa Nizovima i Matricama....12
Funcijsko programiranje u juliji....14
Vizualizacija podataka u programskom paketu Plots....18
Opis problema....18
Opis skupa podataka....18
Implementacija programskog riješenja....19
Programski kod za crtanje stupičastog grafa:....21
Stvaranje grafa sa podacima iz 2021. godine:....22
Prikaz rezultata....23
Opis programskog koda za crtanje linijskog grafa razlike broja žena i muškaraca....24
Računanje minimuma i maksimuma složene funkcije pomoću paketa Optim....26
Opis problema....26
Implementacija programskog rješenja....27

Prikaz Rezultata....28
Primjena jezika Julia u Linearnom programiranju....29
Što je linearna optimizacija....29
Opis problema – problem optimizacije....29
Implementacija problema u programskom jeziku Julia....29
Definiranje optimizacijskog modela....30
Pronalazak optimalnog riješenja....31
Višenitno programiranje u jeziku Julia i mjerenje performansi....32
Opis problema....32
Implementacija funkcije za višenitno računanje sume kvadrata niza....32
Definicija višenitne funkcije....33
Programski kod za mjerenje vremena izvođenja i memorije....34
....35
Ispis dobivenih rezultata....35
Zaključak....36
Literatura....37

Uvod

U današnjem brzom razvoju informacijske tehnologije, odabir odgovarajućeg programskog jezika postaje ključan faktor u učinkovitom rješavanju različitih izazova. Mnogi popularni dinamički jezici nisu dizajnirani s ciljem performansi. Važilo je pravilo da se programi koji zahtjevaju visok stupanj performansi pišu u statičkim jezicima C ili Fortran, čime bi bila smanjena produktivnost programera jer su ti programi su složeniji za izradu. Tek s rastućom potrebom u svakodnevnom radu znanstvenih programera za istovremenom produktivnošću i izvedbom, nastala je potreba za dinamičkim jezikom visokih performansi. [1]

U skladu s tim, ovaj završni rad posvećen je analizi i primjeni programskog jezika Julia u kontekstu kolegija Programske paradigme i jezici. Ovaj rad ima za cilj pružiti sveobuhvatan uvid u jezik Julia, istražujući njegove osnovne karakteristike, sintaksu i semantiku, kako bi se omogućilo jasno razumijevanje njegove primjene.

U prvom dijelu rada dat će se pregled jezika Julia, uključujući povijesnu perspektivu, i motivaciju koja stoji iza njegovog nastanka. Bit će razmatrane osnovne značajke koje čine Juliju izuzetno svestranim jezikom za znanstveno računanje i analizu podataka, s posebnim naglaskom na njegov sustav tipova, performanse te mogućnostima funkcijskog programiranja.

Nakon pregleda jezika, u drugom dijelu rada detaljno će se analizirati osnovne karakteristike jezika Julia. Fokus će biti na tipovima podataka i sustavu tipova koji pružaju temelje za visoke performanse u izvođenju programa. Također, bit će razmotrene funkcionalne značajke jezika i načini na koje one doprinose efikasnom rješavanju problema u Juliji.

Kroz implementaciju odabranih problema u jeziku Julia, treći dio rada fokusirat će se na demonstraciju praktične primjene jezika u vizualizaciji podataka, riješavanju problema linearnog programiranja i paralelnom programiranju.

1 Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A Fresh Approach to Numerical Computing. SIAM Review, 59(1), 68-97.

Pregled programskog jezika Julia

Povijest i razvoj jezika julia

Početci i motivacija

Sve je započelo s grupom entuzijastičnih znanstvenika i programera koji su radili na području matematike, statistike i računalnih znanosti. Ti ljudi su primijetili da postojeći programski jezici nisu uvijek učinkoviti kada je riječ o brzom izvođenju numeričkih operacija i paralelnom računanju (Python, Matlab). Stoga su odlučili stvoriti jezik koji bi zadovoljio potrebe znanstvenika, inženjera i analitičara podataka. [2]

Dizajn jezika

Jedan od ključnih ciljeva dizajna jezika Julia bio je omogućiti jednostavan i efikasan rad s numeričkim podacima.[3] Julia je osmišljena kao jezik koji kombinira visoku razinu apstrakcije s performansama koje se mogu usporediti s jezicima kao što su C i Fortran.

Julia se odlikuje svojom jednostavnom, lako razumljivom ali ekspresivnom sintaksom. Tome doprinosi što Julia podržava dinamičku tipizaciju, što znači da programeri ne moraju eksplicitno deklarirati tipove podataka. To pomaže bržem razvijanju koda.

Open-source filozofija

Jedan od ključnih faktora koji je doprinio uspjehu jezika Julia je njegova open-source priroda. To znači da je jezik besplatan za korištenje, a njegov izvorni kod je otvoren za suradnju i doprinose zajednice. Ovo potiče mnoge entuzijaste da aktivno sudjeluju u razvoju jezika i stvaranju novih alata i biblioteka.

Julia je brzo postala popularan jezik u znanstvenim istraživanjima, posebice u područjima kao što su statistika, matematika, fizika i računalne znanosti najviše zbog brzine izvođenja i mogućnosti optimizacije.

Tijekom godina, programska podrška je narasla te su razvijene su mnoge biblioteke i alati koji omogućuju još širi spektar primjena, uključujući strojno učenje, duboko učenje, analizu podataka i druge.

---

2 Julia Programming Language. (2012, 8. februar). Why We Created Julia. https://julialang.org/blog/2012/02/why-we-created-julia/

3 Julia: dynamism and performance reconciled by design" by Bezanson, J., Chen, J., Chung, B., Karpinski, S., Shah, V. B., Vitek, J., & Zoubritzky, L. (2018) u Proceedings of the ACM Programming Language, 2(OOPSLA), Article 120. https://doi.org/10.1145/3276490

Dakle, Julia je jezik koji je nastao iz potrebe za učinkovitim alatom za znanstveno računanje i analizu podataka. Njen uspjeh leži u kombinaciji brzine izvođenja s jednostavnom sintaksom, ali i zbog brojne i aktivne zajednice korisnika koja je doprinjela njenom razvoju. Danas, se Julia sve više koristi u znastvenom istraživanju i analizi podataka.

Osnovne značajke jezika julia

Brzina izvršavanja i JIT kompajliranje

Jezik Julia se izdvaja po iznimnoj brzini izvršavanja. Koristi tehnologiju Just-In-Time (JIT) kompajliranja koja omogućava efikasnu pretvorbu koda u strojni jezik tijekom izvršavanja programa.

JIT kompajliranje je tehnika koja omogućava dinamičku interpretaciju i optimizaciju koda tijekom izvršavanja programa. Umjesto potpunog prevođenja u strojni kod prije izvršavanja, JIT kompajler prevodi dijelove koda koji se trenutno izvršavaju u strojni jezik.

Multiple dispatch

Ključna karakteristika koja izdvaja jezik Julia u odnosu na druge programske jezike je *multiple dispatch*.

Ono omogućava pisanje funkcija koje se može prilagoditi različitim tipovima podataka. Ova fleksibilnost u definiranju funkcija po tipovima ulaznih argumenata pomaže u izbjegavanju grešaka u kodu i pridonosi jasnoći implementacije i lakoći generičkog programiranja.

Dinamički tipovi s opcionalnim anotacijama tipova

Julia je dinamički tipizirani jezik, što znači da tipovi varijabli se mogu mijenjati i ne moraju se navesti eksplicitno. Opcionalne anotacije tipova su dostupne ako je potrebno pridonijeti performansama ili poboljšati čitljivost koda. Ova značajka omogućava brzo prototipiranje i olakšava suradnju u timskom okruženju.

Za razliku od nekih drugih jezika koji mogu biti sporiji zbog interpretacije koda, Julia postiže visoke performanse zahvaljujući JIT prevođenju izvornog koda u strojni jezik.

Paralelno računanje i distribuirano računarstvo

Julia nudi ugrađene mehanizme za paralelno računanje, omogućujući iskorištavanje više jezgri procesora za brže računanje (koristeći programske niti). Osim toga, ima podršku za distribuirano

računarstvo, što omogućava istovremeni rad više računalnih čvorova na rješavanju složenih problema.

Analiza jezika Julia

Kontrolne strukture

Kontrolne strukture u programskom jeziku Julia su slične većini programskih jezika, koristi iste ključne riječi kao u programskom jeziku Python uz razliku da ne koristi identaciju koda za definiranje programskog bloka već ključnu riječ `end` .

- If-Else

```
if x > 5
    println("x je veći od 5")
elseif x > 0
    println("x je pozitivan, ali manji ili jednak 5")
else
    println("x je negativan")
end
```

- For Petlja:

```
for i in 1:5
    println(i)
end
```

- While Petlja:

```
i = 1
while i <= 5
    println(i)
    i += 1
end
```

• Break i Continue:

break se koristi za izlazak iz petlje, continue preskače trenutnu iteraciju i nastavlja s sljedećom.

```
for i in 1:10
    if i == 5
        break
    end
    println(i)
end
```

Tipovi podataka u juliji

Julia ima kompleksan sustav tipova podataka, koji se sastoji od stabla tipova, svi tipovi podataka pripadaju apstraktnom tipu „Any”.

Vrste tipova podataka

1. Abstraktni tipovi (Abstract Types): Abstraktni tipovi su viši nivoi apstrakcije koji ne mogu imati instance. Oni se koriste za organiziranje i hijerarhijsko grupiranje tipova. Na primjer, Number je apstraktni tip koji uključuje sve numeričke tipove poput Int, Float, itd.
2. Konkretni tipovi (Concrete Types): Konkretni tipovi mogu imati instance i često predstavljaju konkretnu reprezentaciju podataka. Na primjer, Int je konkretni tip koji predstavlja cjelobrojne brojeve.

Stablo tipova:

Julia koristi usmjereni aciklički graf (Directed Acyclic Graph - DAG) za organizaciju tipova. Na vrhu ovog grafa nalaze se najviši apstraktni tipovi, poput Any, koji je super-tip za sve druge tipove u Juliji. Odatle se granaju u više specifične abstraktne tipove, poput Number koji predstavlja sve numeričke tipove, AbstractString za znakovne nizove, AbstractArray za sve tipove nizova, Function za funkcije uključujući anonimne i funkcije višeg reda.
Ispod apstraktnih tipova smještaju se konkretni tipovi koji nasljeđuju svojstva i metode svojih apstraktnih roditelja.

Primjerice, u Juliji tip Float64 je podtip Number jer je Float64 konkretni podtip koji spada pod opći apstraktni pojam brojeva.

Prednosti stabla tipova u juliji:

1. Performanse: Stablo tipova pomaže kompajleru Julije da generira vrlo učinkovit kod. Budući da Julia koristi informacije o tipu tijekom izvođenja, pravilno definirani tipovi omogućuju optimizacije poput specijalizacije i inlininga.
2. Polimorfizam: Julia podržava multipleksni dispečer, što znači da se funkcija može ponašati različito ovisno o tipu ulaznih argumenata. To omogućuje fleksibilnost u programiranju.
3. Fleksibilnost i apstrakcija: Abstraktni tipovi omogućuju programerima da rade na višim razinama apstrakcije, dok konkretni tipovi omogućuju precizno definiranje podataka i operacija.

Osnovni tipovi podataka

1. Integer (Cjelobrojni brojevi): Ovo su brojevi koji nemaju decimalne dijelove. Primjeri uključuju -1, 0, 1, 2, itd. U Juliji, ovo može biti Int32, Int64, UInt32, UInt64, ovisno o veličini memorije koja se koristi.
2. Floating Point (Pomični zarez): Ovi tipovi podataka uključuju brojeve s decimalnim dijelovima. Primjeri uključuju -1.5, 0.0, 3.14, itd. U Juliji, ovo može biti Float16, Float32, Float64 ovisno o preciznosti.
3. Boolean (Logičke vrijednosti): Ovaj tip podataka može biti samo true ili false. Koristi se za kontrolu toka programa i uvjetne izraze.
4. Char (Znakovi): Ovaj tip podataka predstavlja pojedinačne znakove, npr. 'a', 'b', '1'. U Juliji se označava jednostrukim navodnicima.
5. String (Nizovi znakova): Ovo je niz znakova, kao što su "Hello, World!". U Juliji se označava dvostrukim navodnicima.
6. Symbol (Simboli): Simboli su imenovane vrijednosti koje se koriste kao identifikatori. Na primjer, :ime_simbola.

Kolekcije:

1. Array (Polje): Polje je osnovna kolekcija u Juliji koja omogućuje pohranjivanje više elemenata različitih tipova. Na primjer, arr = [1, 2, 3] stvara polje s tri elementa.
2. Tuple (Uredjeni niz): Tuple su slični poljima, ali su nepromjenjivi (immutable), što znači da se nakon stvaranja ne mogu mijenjati. Na primjer, tup = (1, "hello", 3.14).
3. Set (Skup): Skup je kolekcija koja sadrži jedinstvene elemente, bez redoslijeda. Na primjer, my_set = Set([1, 2, 2, 3]) stvara skup {1, 2, 3}.
4. Dictionary (Rječnik): Rječnik omogućuje povezivanje ključeva s vrijednostima. Svaki ključ mora biti jedinstven. Na primjer, my_dict = Dict("kljuc1" => 10, "kljuc2" => 20).

Složeni tipovi podataka:

1. Struct (Struktura): Struktura je način definiranja složenih tipova koji grupiraju različite podatke pod jednu cjelinu. Na primjer, ako želite predstaviti informacije o osobi, možete stvoriti strukturu koja sadrži ime, dob, i adresu.

```
struct Osoba
    ime::String
    dob::Int
    adresa::String
end

moja_osoba = Osoba("Ana", 30, "Ulica Primorska 12")
```

2. Enum (Enumeracija): Enumeracije omogućuju definiranje skupa diskretnih vrijednosti. To može biti korisno za predstavljanje stanja u programu. Na primjer, definiranje enumeracije za dane u tjednu.

```
enum DanUTjednu
    Ponedjeljak
    Utorak
    Srijeda
    Četvrtak
    Petak
    Subota
    Nedjelja
end

danas_je = DanUTjednu.Utorak
```

3. Union (Unija): Unija je tip koji može biti jedan od nekoliko drugih tipova. Koristi se kada želite dopustiti varijabilnost u tipu neke varijable.

```
function foo(x::Union{Int, Float})
    return x * 2
end
```

4. Function (Funkcija): U Juliji, funkcije su također složeni tipovi podataka. To znači da možete dodijeliti funkciju varijablama, prosljeđivati ih kao argumente i vraćati ih kao rezultate.

```
function square(x)
    return x^2
end

my_func = square
result = my_func(5)  # Ovo će biti 25
```

Rad sa Nizovima i Matricama

Nizovi u Juliji su osnovna struktura podataka koja omogućava pohranjivanje više elemenata različitih tipova.

- Stvaranje niza:

```
# Stvaranje jednodimenzionalnog niza
arr1d = [1, 2, 3, 4, 5]

# Stvaranje višedimenzionalnog niza
arr2d = [1 2 3; 4 5 6; 7 8 9]
```

- Pristup elementima niza:

```
# Pristup trećem elementu
elem = arr1d[3]

# Promjena vrijednosti elementa
arr1d[1] = 10  # Promijenit će prvi element u 10
```

• Operacije na nizovima:

```
# Dodavanje elemenata
push!(arr1d, 6)  # Dodaje 6 na kraj niza

# Uklanjanje elemenata
pop!(arr1d)  # Uklanja zadnji element

# Duljina niza
len = length(arr1d)

# Iteriranje kroz niz
for element in arr1d
    println(element)
end
```

• Stvaranje matrice:

```
matrica = [1 2 3; 4 5 6; 7 8 9]
```

• Pristup elementima matrice:

```
# Pristup elementu u 2. retku i 3. stupcu
elem = matrica[2, 3]

# Promjena vrijednosti elementa
matrica[1, 1] = 10  # Promijenit će se prvi element u 10
```

• Operacije na matricama:

```
# Transponiranje matrice
transponirana_matrica = transpose(matrica)

# Množenje matrica
nova_matrica = matrica * transponirana_matrica

# Inverz matrice
inverz_matrice = inv(matrica)

# Determinanta matrice
determinanta = det(matrica)
```

Funcijsko programiranje u juliji

Julia je programski jezik koji podržava funkcijsko programiranje. Ova paradigma programiranja uključuje niz značajki koje olakšavaju pisanje funkcijskog koda. U Juliji možete koristiti lambda funkcije, higher-order funkcije, imutabilnost i rekurziju.

Lambda funkcije su anonimne funkcije koje se mogu koristiti kao argumenti drugih funkcija ili dodijeliti varijablama.

Primjer, lambda funkcije koja računa kvadrat broja:

```
kvadrat = x -> x^2
rezultat = kvadrat(5)
println(rezultat)  # Output: 25
```

Funkcije višeg reda su funkcije koje primaju druge funkcije kao argumente ili vraćaju funkcije kao rezultat. Na primjer, moguće je definirati funkciju višeg reda koja primjenjuje funkciju na svaki element niza:

```
function primijeni_na_sve(funkcija, lista)
    rezultat = []
    for element in lista
        push!(rezultat, funkcija(element))
    end
    return rezultat
end

lista = [1, 2, 3, 4, 5]
kvadrati = primijeni_na_sve(x -> x^2, lista)
println(kvadrati)  # Output: [1, 4, 9, 16, 25]
```

Nepromjenjivost (*immutability*) je koncept koji promiče da se podaci ne mijenjaju izravno, već se stvaraju nove verzije podataka. To olakšava pisanje sigurnog i paralelnog koda.
Struktura je primjer podatka koji se nakon inicijalizacije ne može mijenjati.

```
struct Osoba
    ime::String
    prezime::String
    godine::Int
end

osoba = Osoba("Ana", "Anić", 30)
osoba.godine = 31  # Vraća Grešku
```

Rekurzija je tehnika u kojoj funkcija poziva samu sebe. To je korisno za rješavanje problema koji se mogu prirodno izraziti rekurzivno.
Primjer rekurzivne funkcije koja računa n-ti broj fibonaccijevog niza.

```
function zlatni_rez(n)
    if n == 0
        return 1
    elseif n == 1
        return 1
    else
        return zlatni_rez(n-1) + zlatni_rez(n-2)
    end
end

rezultat = zlatni_rez(7)
println(rezultat)  # Output: 21
```

Funkcije map, filter i reduce

```
# Primjeri korištenja funkcija map filter i reduce:
lista = [1, 2, 3, 4, 5]

# Primjer: Kvadriranje svakog elementa liste

kvadrati = map(x -> x^2, lista)
println(kvadrati)  # Output: [1, 4, 9, 16, 25]

# Primjer: Filtriranje parnih brojeva iz liste

parni = filter(x -> x % 2 == 0, lista)
println(parni)  # Output: [2, 4]

# Primjer: Zbrajanje svih elemenata liste

zbroj = reduce(+, lista)
println(zbroj)  # Output: 15
```

Vizualizacija podataka u programskom paketu Plots

Programski paket Plots je jedan od najpopularnijih paketa za vizualizaciju podataka u jeziku Julia. Omogućava korisnicima da kreiraju različite vrste grafikona, dijagrama i vizualizacija na jednostavan način.

Plots podržava širok spektar grafikona, uključujući linijske grafikone, stubičaste grafikone, tačkaste grafikone, površinske grafikone, histogram, raspodelu verovatnoće, 3D grafikone i mnoge druge.

Opis problema

Cilj je izraditi tri grafa pomoću programskog paketa Plots, koji pokazuju odnos spolova u Republici Hrvatskoj za različite dobne skupine na osnovu podataka sa stranice Zavoda za statistiku Republike Hrvatske iz 2018. i 2021. godine.
Podaci će se učitati iz csv datoteka, filtrirati potom prikazati u dva stupičasta grafa, jedan za 2018. a drugi za 2021. godinu.
Ideja je na vizualan način u dva stupičasta grafa prikazati razlike u broju muškaraca i žena kroz životnu dob.

Za stvaranje trećeg linijskog grafa će se iskoristiti podaci iz oba skupa na način da se za svaki definira funkcija razlike između ženske i muške populacije. Funkcije će se zajedno iscrtati na linijskom grafu.

Opis skupa podataka

U podakovnom skupu iz 2018. godine u izračunima su korišteni retci sa podacima o ukupnoj populaciji, ukupnom broju muškaraca, žena za sve dobne skupine u diskretnim jedinicama od 5 gdje je prva skupina sa godinama 0-4, sljedeća 5-9, sve do zadnje skupine onih sa 85 ili više godina.
U podakovnom skupu iz 2021. godine su retci koji sadrže podatke o spolu za dobne skupine definirane identično kao ranije, uz razliku da su retci i stupci zamjenjeni.

Implementacija programskog riješenja

U prvom koraku se instaliraju potrebni paketi i učitavaju:

```
# Instalacija paketa
using Pkg
Pkg.add("DataFrames")
Pkg.add("Plots")
Pkg.add("CSV")

# Učitavanje paketa
using DataFrames
using Plots
using CSV
```

Zatim se podaci čitaju iz CSV datoteke i spremaju u DataFrame:

```
# Učitavanje podataka iz CSV datoteke
podaci_2018 = DataFrame(CSV.File("2018Croatia.csv"))
podaci_2021 = DataFrame(CSV.File("2021Croatia.csv"))
```

Slijedi filtriranje podataka iz podakovnog skupa `popis_2018` gdje se uzimaju podaci za sve životne dobi.

Operatorom sa parametrima `[1:end, 1:4]` se iz varijable `hrvatska` uzimaju svi retci (sve dobne skupine), i prva 4 stupca (Ukupna populacija, broj muškaraca i broj žena).

Pošto prvi stupac u varijabli `hrvatska` sadrži podatak o ukupnoj populaciji, koji izračunu nije potreban (cilj je pokazati razlike u broju muškaraca i žena prema životnoj dobi) pa se uzimaju stupci sa brojem muškaraca i žena koristeći operator uglatih zagrada `[2:end]`.

```
# Učitavanje broja ljudi, muškaraca i žena svih dobi
hrvatska = podaci_2018[1:end, 1:4]

dobne_skupine = hrvatska.Age[2:end]
muska_populacija = hrvatska[!, Symbol("2018 Republic of Croatia Men")][2:end]
zenska_populacija = hrvatska[!, Symbol("2018 Republic of Croatia Women")][2:end]
```

Sada varijable sadrže jednodimenzionalne vektore čija pozicija u nizu označava dobnu skupinu (uzlazno), koji će se koristiti za crtanje stupičastog grafa.

Programski kod za crtanje stupičastog grafa:

```
# crtanje grafa
Plots.bar(
    dobne_skupine,
    [muska_populacija, zenska_populacija],
    xlabel="Dobne skupine", ylabel="Populacija",
    label=["Surplus muškaraca" "Surplus žena"],
    legend=:topleft,
     title="Odnos broja muškaraca i žena kroz životnu dob za 2018 godinu.",
    color = [:blue :red], alpha = [1 0.5])

Plots.savefig("hrvatska2018.png")
```

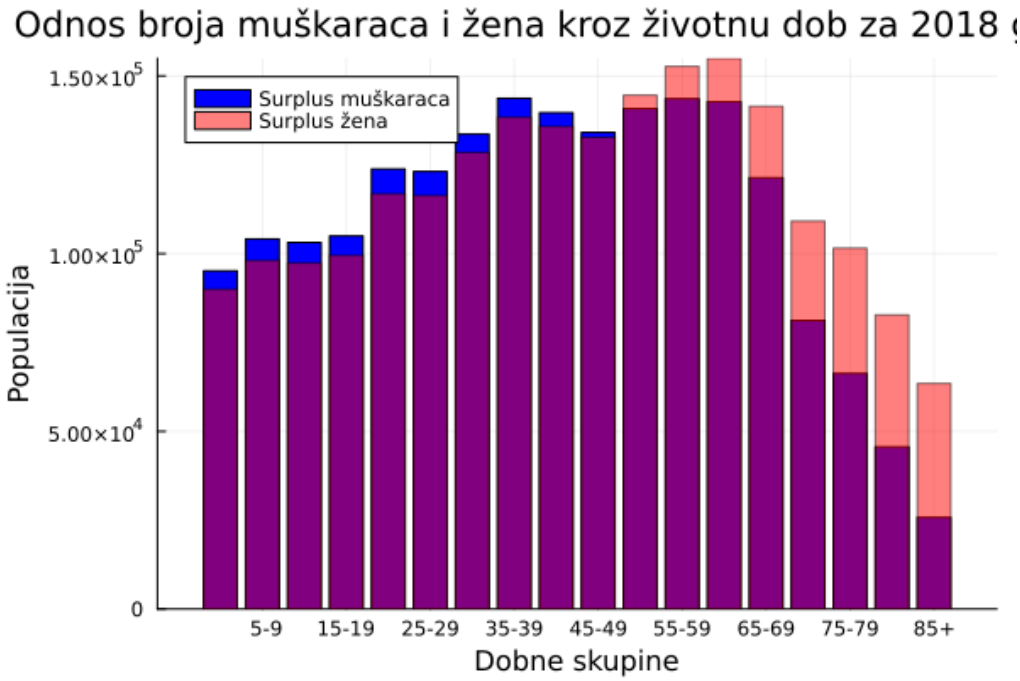
Koristeći vektore iz prethodnog koraka pomoću funkcije `Plots.bar()` stvara se graf. Prva dva argumenta pružaju vrijednosti za x i y kordinatnu os dok ostali parametri određuju izgled grafa. Naposlijetku se graf sprema kao slika u datoteku.

Stvaranje grafa sa podacima iz 2021. godine:

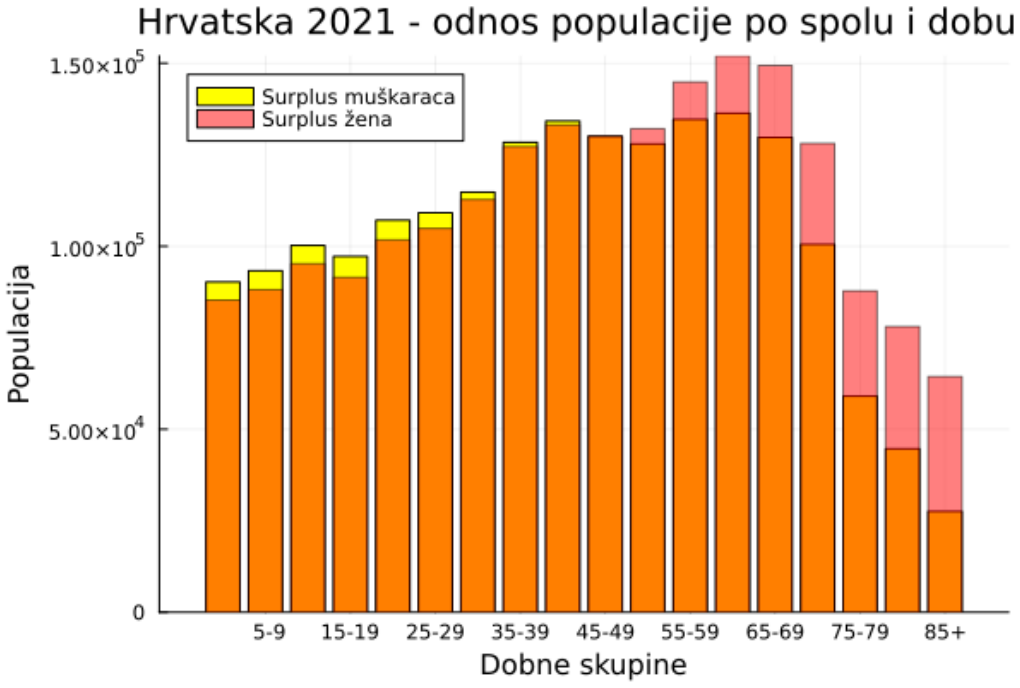
```
ukupna_populacija_2021 = Vector(first(podaci_2021)[3:end])
muska_populacija_2021 = Vector(podaci_2021[2, :][3:end])
zenska_populacija_2021 = Vector(podaci_2021[3, :][3:end])

# crtanje grafa 2021
Plots.bar(
    dobne_skupine,
    [muska_populacija_2021, zenska_populacija_2021],
    xlabel="Dobne skupine", ylabel="Populacija",
    label=["Surplus muškaraca" "Surplus žena"],
    legend=:topleft,
    title="Hrvatska 2021 - odnos populacije po spolu i dobu ",
    color = [:yellow :red], alpha = [1 0.5])

Plots.savefig("hrvatska2021.png")
```

Prikaz rezultata

Slika 1. Stupičasti graf za 2018. godinu



Slika 2. Stupičasti graf za 2021. godinu



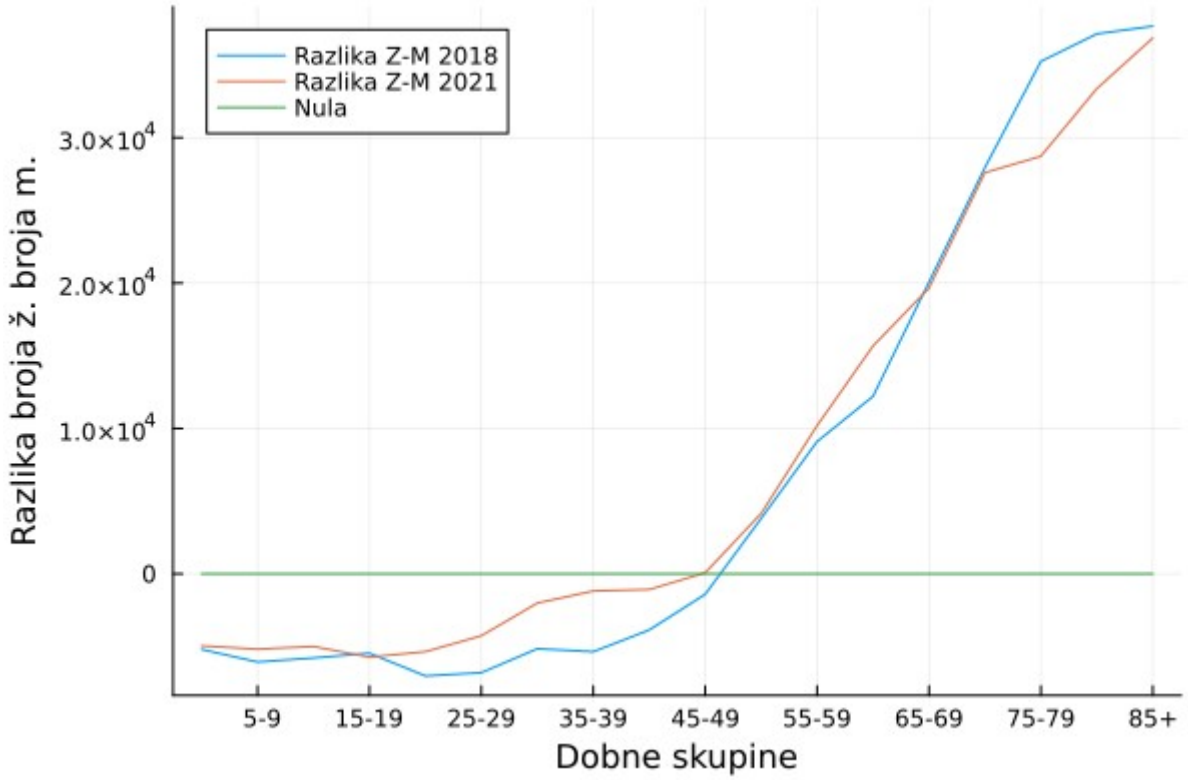
Opis programskog koda za crtanje linijskog grafa razlike broja žena i muškaraca

Pomoću funkcije višeg reda se oduzmu brojevi populacije žena i muškaraca za svaku dob i dobije se novi vektor koji sadrži razliku populacija pojedinačnih dobnih skupina.
Taj novi vektor prestavlja vrijednosti za y - os, a za x - os se uzimaja vektor `dobne_skupine`.

Za crtanje linijskog grafa poziva se funkcija `Plots.plot()` koja kao prva dva parametra prima vektore za x – os, odnosno y – os.

```
razlika(x, y) = x - y

razlike_m_z = map(razlika, zenska_populacija,    muska_populacija)
razlike_m_z_2021     =     map(razlika,     zenska_populacija_2021,
muska_populacija_2021)

Plots.plot(dobne_skupine,
     razlike_m_z,
     label="Razlika Z-M 2018",
     xlabel="Dobne skupine",
     ylabel="Razlika broja ž. broja m.")


plot!(dobne_skupine, razlike_m_z_2021, label="Razlika Z-M 2021")
plot!(dobne_skupine, x -> 0, label="Nula" )

Plots.savefig("linijski_graf")
```

Slika 3. Razlika populacija žena i muškaraca po životnoj dobi



Računanje minimuma i maksimuma složene funkcije pomoću paketa Optim

Opis problema

Definirane su sljedeće funkcije:

- f(x) = (x - 2)^2
- g(x) = (x - 1)^2
- h(x) = f(x) + g(x)

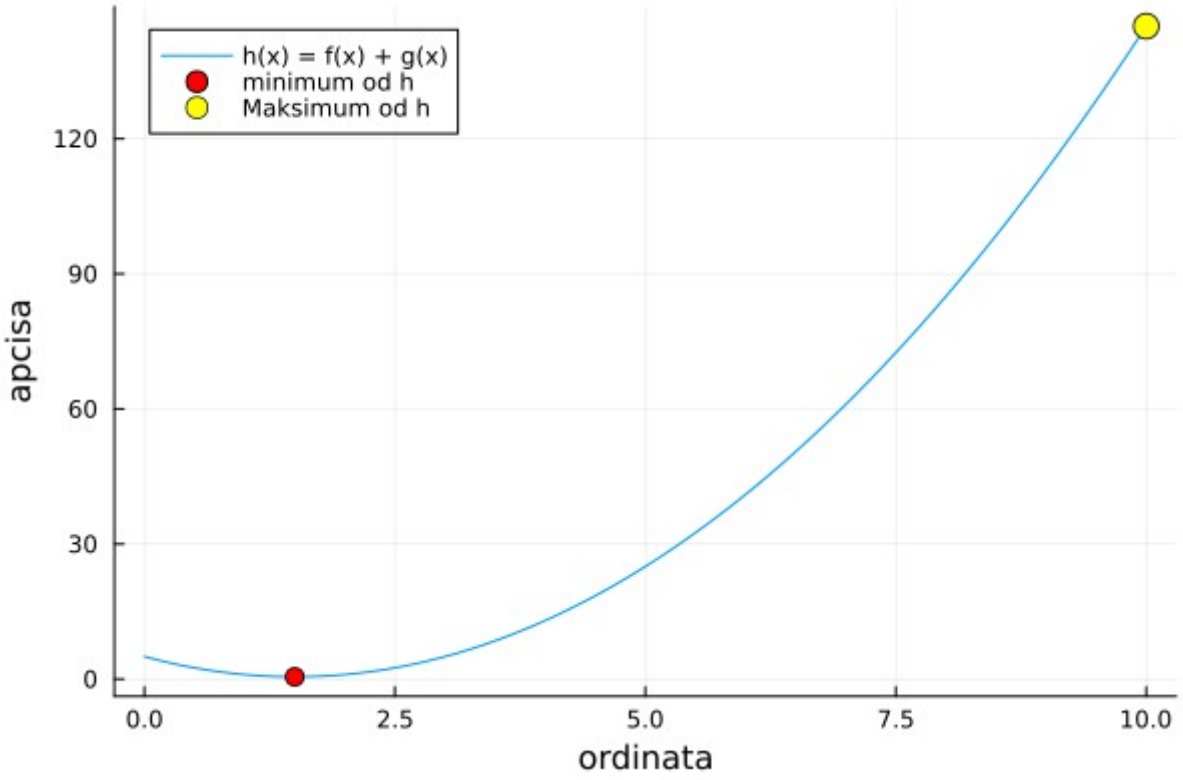
Cilj je pronaći vrijednosti od x za koju funkcija h(x) postiže maksimalnu i minimalnu vrijednost na zadanom rasponu, nacrtati graf funkcije h i označiti točke maksimuma i minimuma.

Implementacija programskog rješenja

```
using Optim
using Plots

f(x) = (x - 2)^2
g(x) = (x - 1)^2
h(x) = f(x) + g(x)

rezultat = optimize(h, 0.0, 10.0)
REZULTAT = maximize(h, 0.0, 10.0)

# Računanje minimuma
najmanja_vrijednost = Optim.minimum(rezultat)
najmanja_vrijednost_argumenta = Optim.minimizer(rezultat)

# Računanje Maksimuma
Najveca_vrijednost = Optim.maximum(REZULTAT)
Najveca_vrijednost_argumenta = Optim.maximizer(REZULTAT)

X_MAX = [Najveca_vrijednost_argumenta]
Y_MAX = [Najveca_vrijednost]
x_min = [najmanja_vrijednost_argumenta]
y_min = [najmanja_vrijednost]
eksevi = 0.0:0.01:10.0
epsiloni = map(h, eksevi)

Plots.plot(eksevi, epsiloni, label="h(x) = f(x) + g(x)",
    xlabel="ordinata", ylabel="apcisa")
    scatter!(x_min, y_min, markersize=5, markercolor=:red, label="minimum od h")
    scatter!(X_MAX, Y_MAX, markersize=7, markercolor=:yellow, label="Maksimum od
h")

Plots.savefig("minMAX")
```

Prikaz Rezultata

Slika 4. Prikaz minimuma i maksimuma funkcije h(x) u rasponu 0 - 10



Primjena jezika Julia u Linearnom programiranju

Što je linearna optimizacija

Linearna optimizacija, također poznata kao linearno programiranje (LP), matematička je metoda za pronalaženje najboljeg ishoda u modelu koji se sastoji od linearnih odnosa. Uključuje optimizaciju (maksimiziranje ili minimiziranje) funkcije linearnog cilja, koja podliježe skupu ograničenja linearne jednakosti i nejednakosti.

Opis problema – problem optimizacije

Seljak ima 20 hektara za uzgoj ječma i pšenice. On mora odlučiti koliko će posijati jednog i drugog. Trošak po hektaru za ječam iznosi 30 eura, a za pšenicu 20 eura. U budžetu ima 480 eura. Ječam zahtjeva 1 radni dan, a pšenica 2 radna dana, ukupno ima 36 radnih dana. Profit na ječmu iznosi 100 eura po hektaru, a na pšenici 120 eura. Koliko treba posijati hektara ječma a koliko hektara pšenice da bi najviše zaradio ?

Implementacija problema u programskom jeziku Julia

Programski jezik Julia omogućuje jednostavno riješavanje problema linearnog programiranja pomoću gotovih solvera i modela.

```
# Najprije instaliramo potrebne pakete, JuMP za modeliranje, GLPRK za računanje

using Pkg; Pkg.add(JuMP);Pkg.add(GLPRK)
using JuMP, GLPK

# priprema optimizacijskog modela
m = Model(GLPK.Optimizer)
```

Varijable odlučivanja su `broj_hektara_jecma` i `broj_hektara_psenice`, a ograničenja su sljedeća:

- `broj_hektara_jecma + broj_hektara_psenice < = 20`
- `broj_hektara_jecma + 2 * broj_hektara_psenice < = 36`
- `30 * broj_hektara_jecma + 20 * broj_hektara_psenice < = 480`

Funkcija cilja je `100 *  broj_hektara_jecma + 120 * broj_hektara_psenice`

Definiranje optimizacijskog modela

```
# var odlučivanja
@variable(m, broj_hektara_jecma >=0)
@variable(m, broj_hektara_psenice >=0)

# ograničenja
@constraint(m, ogranicenje1, broj_hektara_jecma +
     broj_hektara_psenice < = 20)
@constraint(m, ogranicenje2, broj_hektara_jecma + 2 *
     broj_hektara_psenice < = 36)
@constraint(m, ogranicenje3,   30  *  broj_hektara_jecma  +  20  *
     broj_hektara_psenice < = 480)

# funkcija cilja
@objective(m, Max, 100 *  broj_hektara_jecma + 120 *
     broj_hektara_psenice)
```

Pronalazak optimalnog riješenja

```
# izračunava optimalno riješenje
JuMP.optimize!(m)

println("Optimalna rješenja:")
println("broj_hektara_jecma ”,
     JuMP.value(broj_hektara_jecma))
println("broj_hektara_psenice ",
     JuMP.value(broj_hektara_psenice))

# Ispis:
#
# Optimalna rješenja:
# broj_hektara_jecma 4.0
# broj_hektara_psenice 16.0
#
# Objašnjenje: Seljak treba posijati 4 hektara ječma i 16 hektara
# pšenice
```

Višenitno programiranje u jeziku Julia i mjerenje performansi

Opis problema

Cilj je pokazati primjenu Julie za višenitno programiranje, na način da se implementiraju dvije funkcije koje će računati sumu kvadrata svakog elementa niza. Jedna od funkcija će se izvršavati u više programskih niti a druga funkcija će se izvršavati normalno.
Zatim će se mjeriti vrijeme izvođenja i iskorištena memorija za obe funkcije.

Implementacija funkcije za višenitno računanje sume kvadrata niza

Najprije se uvezu potrebne biblioteke i definira pomoćna funkcija koju će koristiti višenitna glavna funkcija.
Uloga pomoćne funkcije se svodi na računanje sume kvadrata za djelove niza u svakoj od programskih niti (threadova) .

Definicija višenitne funkcije

```
# računanje zadanog dijela sume kvadrata niza
function djelimicna_suma_kvadrata(niz, start, stop)
    dio_sume = 0
    for i = start:stop
        dio_sume += niz[i]^2
    end
    return dio_sume
end

# višenitna funkcija za računanje sume kvadrata niza
function višenitna_suma_kvadrata(niz, broj_tredova)
    len = length(niz)
    veličina_bloka = Int(ceil(len / broj_tredova))
    dijelovi_sume = zeros(Float64, broj_tredova)

    Threads.@threads for i = 1:broj_tredova
        start_idx = (i - 1) * veličina_bloka + 1
        stop_idx = min(i * veličina_bloka, len)
                dijelovi_sume[i]  =  djelimicna_suma_kvadrata(niz,
start_idx, stop_idx)
    end

    return sum(dijelovi_sume)
end

function obicna_suma_kvadrata(niz)
    return sum(x -> x^2, niz)
end
```

Programski kod za mjerenje vremena izvođenja i memorije

```
niz = rand(1:100, 1000)
veliki_niz = rand(1:100, 100000000)
br_niti = 4

# Mjerenje performansi na nizu sa 1000 elemenata
println("Visenitna: ")
@time rezultat_visenitne = višenitna_suma_kvadrata(niz, br_niti)

println("Obicna: ")
@time rezultat_obicne = obicna_suma_kvadrata(niz)

println("suma kv. - visenitna: ", rezultat_visenitne)
println("suma kv. - obicna: ", rezultat_obicne)



# Mjerenje performansi na nizu sa 100 000 000 elemenata
println("\n\nVisenitna: ")
@time rezultat_visenitne = višenitna_suma_kvadrata(veliki_niz, br_niti)

println("Obicna: ")
@time rezultat_obicne = obicna_suma_kvadrata(veliki_niz)

println("suma kv. - visenitna: ", rezultat_visenitne)
println("suma kv. - obicna: ", rezultat_obicne)
```

Ispis dobivenih rezultata

```
Mjerenje performansi na nizu sa 1000 elemenata
Visenitna:
    0.165367 seconds (86.20 k allocations: 6.371 MiB, 99.90% compilation time)
Obicna:
    0.060267 seconds (27.40 k allocations: 1.872 MiB, 99.93% compilation time)
suma kv. - visenitna: 3.319548e6
suma kv. - obicna: 3319548

Mjerenje performansi na nizu sa 100 000 000 elemenata
Visenitna:
  0.091251 seconds (9 allocations: 784 bytes)
Obicna:
  0.094237 seconds (1 allocation: 16 bytes)
rezultat - visenitna: 3.38340334456e11
suma kv. - obicna: 338340334456
```

Zaključak

U ovom završnom radu detaljno smo istražili programski jezik Julia, njegovu povijest, dizajn, osnovne značajke i prednosti.

Analizirali smo tipove podataka u Juliji, uključujući osnovne tipove, kolekcije i složene tipove. Proučili smo rad s nizovima i matricama te istražili koncepte funkcionalnog programiranja u jeziku Julia.

Dalje, istražili smo mogućnosti vizualizacije podataka pomoću programskog paketa Plots, na način da smo nacrtali stupčasti i linijski graf.

U radu smo također primijenili Juliju za rješavanje problema maksimizacije iz linearne optimizacije, pružajući uvid u definiranje optimizacijskih modela i pronalaženje optimalnih rješenja pomoću solvera.

Nadalje, istražili smo višenitno programiranje u jeziku Julia, implementirali funkcije koje koriste višenitnost i usporedili performanse višenitne funkcije sa serijskom.

Zaključno, ovaj rad je pružio sveobuhvatan pregled programskog jezika Julia i njegove primjene u različitim područjima računarstva, od analize podataka, linearnog programiranja do višenitnog programiranja. Julia se pokazala kao svestran i performantan jezik koji nudi brojne prednosti za inženjere i istraživače. S obzirom na njegovu rastuću zajednicu i kontinuirani razvoj, možemo očekivati da će Julia i dalje imati značajnu ulogu u svijetu računarstva i znanosti.

Kroz ovaj rad, nadamo se da smo čitateljima pružili dublje razumijevanje jezika Julia i potaknuli ih da istraže njegove mogućnosti u vlastitim projektima i istraživanjima.

Literatura

1. Kaggle. (2018). Croatia Bureau of Statistics 2018 Population. https://www.kaggle.com/datasets/marins/croatia-bureau-of-statistics-2018-population
2. JuliaLang. (2021). Julia Documentation. https://docs.julialang.org/en/v1/
3. University of Cincinnati Libraries. (n.d.). Julia. https://guides.libraries.uc.edu/julia
4. Official JuliaLang YouTube Channel. (n.d.). https://www.youtube.com/c/julialangofficial
5. JuliaLang Forum. (n.d.). https://discourse.julialang.org/
6. JuliaCon. (n.d.). https://juliacon.org/
7. Državni zavod za statistiku Republike Hrvatske. (n.d.). https://dzs.gov.hr/
8. Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A Fresh Approach to Numerical Computing. SIAM Review, 59(1), 68-97.
9. A. Sengupta (2016). Julia High Performance. Packt Publishing Ltd.
10. "Julia: dynamism and performance reconciled by design" by Bezanson, J., Chen, J., Chung, B., Karpinski, S., Shah, V. B., Vitek, J., & Zoubritzky, L. (2018) u Proceedings of the ACM Programming Language, 2(OOPSLA), Article 120. https://doi.org/10.1145/3276490