# Izrada programskog alata za implementaciju simpleks metode

**Belcar, Petar**

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ORGANIZACIJE I INFORMATIKE**
**VARAŽDIN**

**Petar Belcar**

# IZRADA PROGRAMSKOG ALATA ZA IMPLEMENTACIJU SIMPLEKS METODE

**ZAVRNI RAD**

**Varaždin, 2023.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Petar Belcar**

**Matični broj: 0119041984**

**Studij: Informacijski sustavi**

# IZRADA PROGRAMSKOG ALATA ZA IMPLEMENTACIJU SIMPLEKS METODE

**ZAVRNI RAD**

**Mentor :**

Dr. sc. Nenad Perši

**Varaždin, rujan 2023.**

*Petar Belcar*

**Izjava o izvornosti**

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

*Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi*

---

**Sažetak**

Završni projekt prolazi kroz teoriju linearnog programiranja koja omogućuje funkcioniranje simpleks metode te teoriju same simpleks metode potrebnu kako bi se simpleks metoda mogla programski implementirati. Također sadrži dokumentaciju programske implementacije simpleks metode te korisničku dokumentaciju same implementacije.

**Ključne riječi**: simpleks metoda; linearno programiranje; optimizacija

# Sadržaj

# 1. Uvod

Drugi svjetski rat je otkrio nedostatke u djelovanju organizacija, ponajviše u procesima odlučivanja raspodjele resursa, kapitala i ljudi. U cilju poboljšanja tih kvantitativnih problema, javlja se linearno programiranje.

U sklopu ovog završnog rada proći ćemo što je linearno programiranje, što je problem linearnog programiranja i kako ga formulirati, teoriju koja objašnjava kako linearno programiranje funkcionira i na koje probleme se može primjenjivati. Kao jednu od metoda linearnog programiranja istaknut ćemo simpleks metodu. Proći ćemo teoriju koju obuhvaća te vodič provođenja simpleks metode čiji će koraci biti implementirani u samom programu.

Završni rad će također sadržavati specifičnosti implementacije, konceptualni model program, programsku dokumentaciju po pojedinim fazama i korisničku dokumentaciju koja će objasniti kako krajnji korisnici mogu primijeniti program u svoje svrhe.

Cilj završnog rada je programska implementacija matematičke teorije takva da bi korisnici s minimalnim predznanjem mogli formulirati i naći najbolje rješenje za linearne programe ili barem saznati da najbolje rješenje ne postoji.

# 2. Linearno programiranje

Linearno programiranje je metoda za dobivanje najboljeg rješenja za skup problema koje zovemo problemi linearnog programiranja ili linearni programi. Ti problemi se sastoje od funkcije cilja i ograničenja.

Funkcija cilja je linearna funkcija koju pokušavamo maksimizirati ili minimizirati ovisno o kakvom problemu je riječ.

$$f : \mathbb{R}^n \to \mathbb{R} \tag{2.1}$$

Slobodne varijable i vrijednost funkcije cilja su nam iz $\mathbb{R}$ jer želimo imati mogućnost povećavanja i smanjenja slobodnih varijabla za proizvoljno male veličine koje dovode to proizvoljno malih promjena u vrijednosti funkcije cilja. Ovo ima utjecaja na kakve vrste problema je smisleno primjenjivati linearno programiranje jer moguće da najbolje rješenje koje dobijemo neće isključivo sadržavati cijele brojeve.

Funkcija se sastoji od $n$ cijena $c_i$, koje će biti konstantne, i $n$ slobodnih varijabla $x_i$ čija će se vrijednost mijenjati kako bismo dobili najveću ili najmanju vrijednost funkcije cilja. Funkciju cilja ćemo označavati sa $Z$.

$$\begin{gathered} Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ c_i \in \mathbb{R}, \quad i \in [1, n] \end{gathered} \tag{2.2}$$

Ponekad će nam biti lakše baratati s cijenama i slobodnim varijablama kao vektorima pa definiramo vektore $C$ i $X$ kao

$$\begin{gathered} C = \begin{bmatrix} c_1 & c_2 & \dots & c_n \end{bmatrix} \\ X = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \\ Z = CX \end{gathered} \tag{2.3}$$

Ograničenja problema linearnog programiranja sastoje se od $m$ linearnih jednakosti i nejednakosti koje se sastoje od konstanti $a_{ij}$ i $bi$ te slobodnih varijabli $x_j$.

$$\begin{gathered} a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n = b_i \\ i \in [1, m] \\ a_{ij} \in \mathbb{R}, \quad \forall i, j \end{gathered} \tag{2.4}$$

Nadalje ćemo pretpostaviti da je broj slobodnih varijabla $n$ veći od broja ograničenja koja nisu ograničenja nenegativnosti ili nepozitivnosti $m$.

Ponekad nam je lakše baratati sa $a_{ij}$-ovima i $b_i$-ovima u obliku matrica i vektora pa definiramo

$$\begin{gathered} A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & & & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \\ B = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{bmatrix} \\ AX = B \end{gathered} \tag{2.5}$$

Također ćemo stupce matrice A zvati $a_i$-evima i pretpostaviti da je rang matrice $A$ jednak $m$.

Rješenje za linearni program je bilo koji vektor $X$ te on može ali ne mora zadovoljavati ograničenja. U slučaju da rješenje zadovoljava ograničenja zovemo ga mogućim rješenjem te ćemo se nadalje baviti isključivo takvim riješenima i zvat ćemo ih samo rješenjima. Linearni program ne mora nužno imati ograničenja ali takvi linearni programi nisu pretjerano korisni pa se njima nećemo baviti.

Postoje dvije vrste linearnih programa: opći i standardni.

Opća vrsta problema kako ga definirana Bertsimas i Tsitsikliscite[1] je problem minimalizacije ili maksimalizacije čija ograničenja mogu biti jednakosti ili nejednakosti, te slobodne varijable mogu biti nenegativne ili nepozitivne.

$$\begin{gathered} max.\ ili\ min.\ Z = c_1x_1 + \dots + c_nx_n \\ a_{e1}x_1 + \dots + a_{en}x_n = b_e \\ \vdots \\ a_{f1}x_1 + \dots + a_{fn}x_n \leq b_f \\ \vdots \\ a_{g1}x_1 + \dots + a_{gn}x_n \geq b_g \\ x_i \leq 0, \quad x_j \geq 0, \quad i \neq j \end{gathered} \tag{2.6}$$

Standardi problem kako ga definiraju Bertsimas i Tsitsiklis[1] ima funkciju cilja čiju vrijednost želimo minimizirati, ograničenja koja su jednakosti te slobodne varijable koje su nenegativne.

$$
\begin{aligned}
min. \quad & Z = c_1 x_1 + \cdots + c_n x_n \\
& a_{11} x_1 + \cdots + a_{1n} x_n = b_1 \\
& \vdots \\
& a_{m1} x_1 + \cdots + a_{mn} x_n = b_m \\
& x_i \geq 0, \quad \forall i \in [1, n]
\end{aligned}
\tag{2.7}
$$

Moguće je opći problem minimalizacije pretvoriti u standardi problem. Promotrimo opći problem

$$
\begin{aligned}
min. \quad & Z = c_1 x_1 + c_2 x_2 \\
& a_{11} x_1 + a_{12} x_2 = b_1 \\
& a_{21} x_1 + a_{22} x_2 \leq b_2 \\
& a_{31} x_1 + a_{32} x_2 \geq b_3 \\
& x_1 \geq 0 \\
& x_2 \leq 0
\end{aligned}
\tag{2.8}
$$

Ako želimo pronaći ekvivalenti standardni problem za ovaj opći problem moramo sva ograničenja pretvoriti u jednakosti i osigurati da su nam sve slobodne varijable nenegativne. Kako bismo pretvorili drugo ograničenje u jednakost moramo uvesti novu slobodnu varijablu $x_3$ čiju vrijednost definiramo kao

$$
x_3 = b_2 - (a_{21} x_1 + a_{22} x_2) \tag{2.9}
$$

Možemo na sličan način pretvoriti treće ograničenje u jednakost tako da uvedemo slobodnu varijablu $x_4$ čiju vrijednost definiramo kao

$$
x_4 = (a_{31} x_1 + a_{32} x_2) - b_3 \tag{2.10}
$$

Tako definirani $x_3$ je nužno nenegativan jer znamo da je desna strana nejednakosti nužno veća ili jednaka lijevoj strani. Također znamo da je $x_4$ nenegativan jer je lijeva strana nejednakosti veća ili jednaka desnoj strani. Tako dodane slobodne varijable će imati pripadajuću cijenu koja je jednaka nuli, te će svi osim jednog $a_{i3}$ i $a_{i4}$ biti jednak nuli. $a_{23}$ će biti jednak $1$, dok će $a_{34}$ biti jednak $-1$. Tada imamo linearni problem koji izgleda kao

$$
\begin{aligned}
min. \quad & Z = c_1x_1 + c_2x_2 \\
& a_{11}x_1 + a_{12}x_2 = b_1 \\
& a_{21}x_1 + a_{22}x_2 + x_3 = b_2 \\
& a_{31}x_1 + a_{32}x_2 - x_4 = b_3 \\
& x_1, x_3, x_4 \geq 0 \\
& x_2 \leq 0
\end{aligned}
\tag{2.11}
$$

Slobodne varijable u standardom problemu moraju biti nenegativne, ali varijabla $x_2$ je nepozitivna. Možemo uvesti $x_2 = -x_2'$. Ako bismo supstituirali $x_2$ za $-x_2$ tada bi imali negativne predznake u funkciji cilja i ograničenjima, no možemo definirati cijenu $c_2 = -c_2'$ i $a_{i2} = -a_{i2}'$. Također bi imali negativan predznak u drugom ograničenju uz varijablu $x_2$ tada možemo uvesti $a_{34} = -1$. Tada imamo standardi linearni problem

$$
\begin{aligned}
min. \quad & Z = c_1x_1 + c_2'x_2' \\
& a_{11}x_1 + a_{12}'x_2' = b_1 \\
& a_{21}x_1 + a_{22}'x_2' + x_3 = b_2 \\
& a_{31}x_1 + a_{32}'x_2' + a_{34}x_4 = b_3 \\
& x_1, x_3, x_4, x_2' \geq 0
\end{aligned}
\tag{2.12}
$$

Rješenje $X'$ za tako dobiveni standardni problem nije rješenje za linearne probleme iz kojeg smo dobili standardi problem jer on ima više slobodnih varijabli. Rješenje $X$ za originalni linearni problem možemo dobiti tako da iz $X'$ izbacimo sve osim prve dvije vrijednosti.

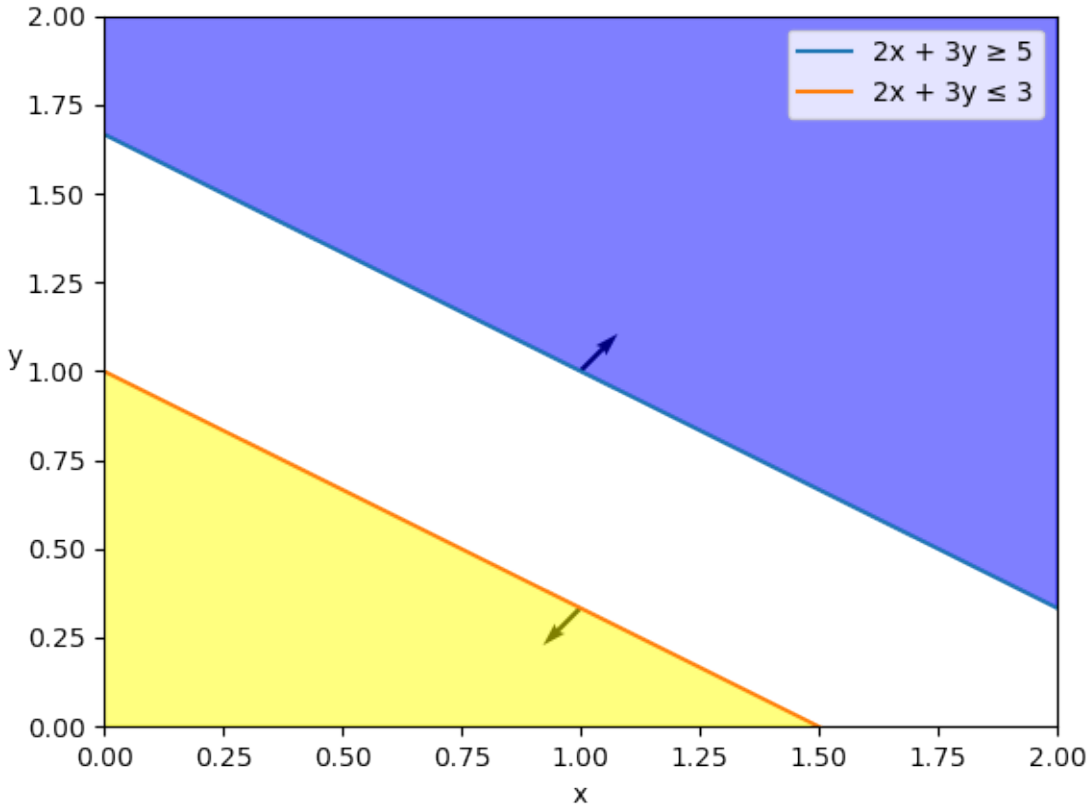
Nadalje kada promatram neki linearni program on će biti standardi linearni program osim ako vrsta programa nije posebno spomenuta.

## 2.1. Rješenja linearnih programa

Svaki linearni program ima skup rješenja koji ćemo zvati $S$ te je broj rješenja u skupu ovisan o ograničenjima. Linearni program ne mora imati rješenja, ali u slučaju da ima rješenja može imati jedno ili beskonačan broj rješenja. Iz skupa rješenja možemo izdvojiti rješenja koja za linearni program maksimalizacije postiže najveću vrijednost funkcije cilja ili najmanju za linearni program minimalizacije. Takva rješenja se zovu optimalnim te linearni program koji ima rješenja ne mora imati optimalna rješenja.

Promotrimo tada 4 primjera linearnih programa koje ćemo prikazati na koordinatnom sustavu na kojem će apscisa imati vrijednost slobodne varijable $x$, a ordinata vrijednosti slobodne varijable $y$.

$$\begin{aligned} max. \quad & Z = x + y \\ & 2x + 3y \geq 5 \\ & 2x + 3y \leq 3 \\ & x, y \geq 0 \end{aligned} \tag{2.13}$$



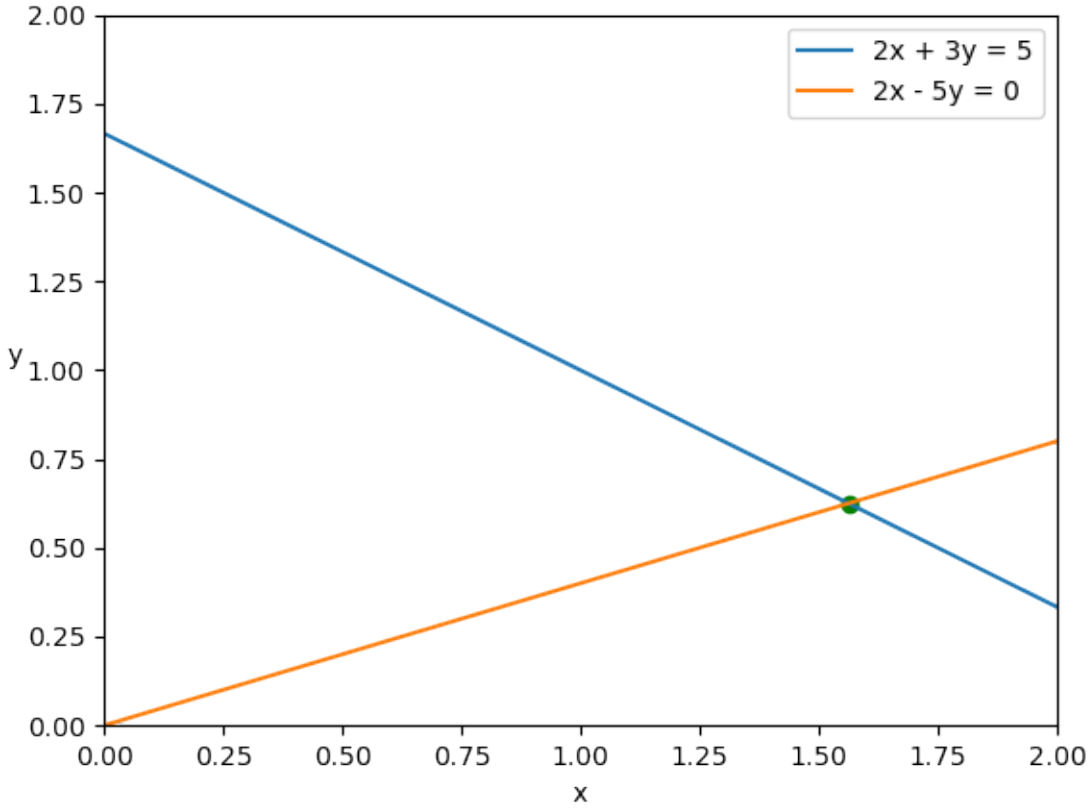

Slika 1: Prikaz ograničenja linearnog programa 2.13

U prvom primjeru iz samih ograničenja možemo zaključiti da nećemo moći naći $x$ i $y$

takve da zadovoljavaju sva ograničenja. Na koordinatnom sustavu vidimo dva pravca svaki od kojih prikazuje jedno od ograničenja koja nisu ograničenja nenegativnosti. Na svakom pravcu se nalazi strelica koja prikazuje na kojoj strani pravca vrijedi nejednakosti, također je taj dio koordinatnog sustava za oba pravca obojan. Vidimo da te dvije površine nemaju presjek te možemo zaključiti da je $S$ za ovaj linearni program prazan.

Promotrimo linearni program koji ima samo jedno rješenje

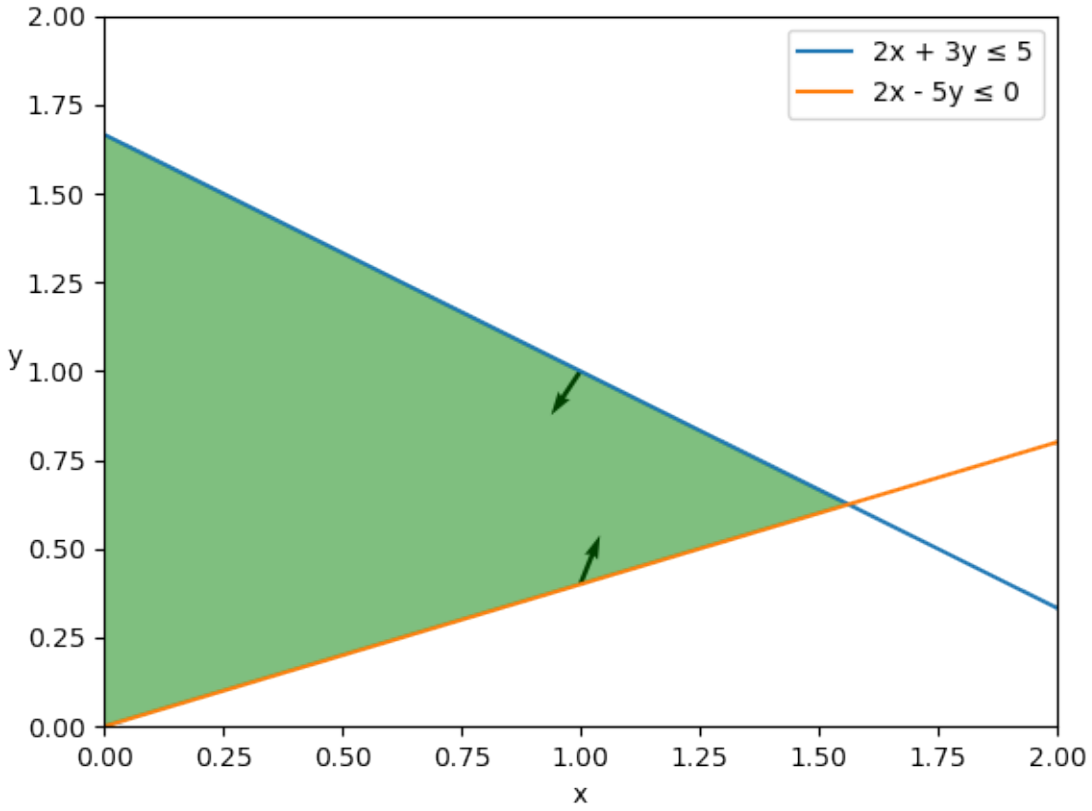
$$\begin{aligned} max. \quad & Z = x + y \\ & 2x + 3y = 5 \\ & 2x - 5y = 0 \\ & x, y \geq 0 \end{aligned} \tag{2.14}$$




Slika 2: Prikaz ograničenja linearnog programa 2.14

Za ovaj slučaj imamo dvije jednakosti i dvije slobodne varijable te bi očekivali da ako postoji rješenje da bi ono bilo cijeli pravac ili jedna točka. U ovom slučaju je jedna točka, ali u slučaju da se pravci preklapaju imamo beskonačni broj rješenja tada može biti riječ o omeđenom problemu. Promotrimo jedan takav primjer.

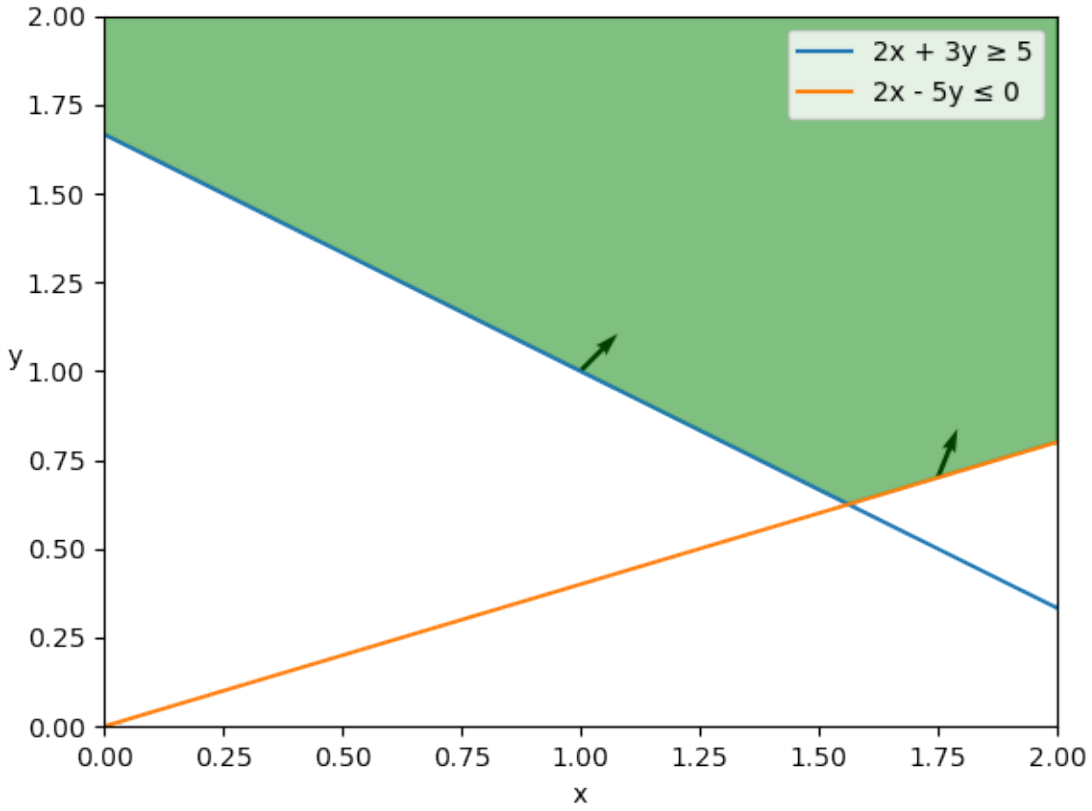
$$
\begin{aligned}
max. \quad & Z = x + y \\
& 2x + 3y \leq 5 \\
& 2x - 5y \leq 0 \\
& x, y \geq 0
\end{aligned}
\tag{2.15}
$$




Slika 3: Prikaz ograničenja linearnog programa 2.15

Zeleno obojani oblik se sastoji od skupa rješenja za linearni problem 2.15. Pošto su rješenja iz $\mathbb{R}^2$ znamo da linearni program ima beskonačan broj rješenja, ali $x$ i $y$ mogu poprimiti vrijednosti samo iz omeđenog skupa brojeva. Ovo nije slučaj za zadnji primjer.

$$
\begin{aligned}
max. \quad & Z = x + y \\
& 2x + 3y \leq 5 \\
& 2x - 5y \leq 0 \\
& x, y \geq 0
\end{aligned}
\tag{2.16}
$$




Slika 4: Prikaz ograničenja linearnog programa 2.16

Vidimo da $x$ i $y$ mogu poprimiti vrijednosti iz $[a, +\infty\rangle, \ \ a \in \mathbb{R}$ te vrijednost funkcije cilja raste linearno s $x$ i $y$ te vrijedi

$$
\begin{aligned}
\forall \begin{bmatrix} x \\ y \end{bmatrix} \in S, \ \exists \begin{bmatrix} x' \\ y' \end{bmatrix} \in S \\
C \begin{bmatrix} x \\ y \end{bmatrix} < C \begin{bmatrix} x' \\ y' \end{bmatrix}
\end{aligned}
\tag{2.17}
$$

Ovakvi linearni programi su neomeđeni i za njih nije moguće naći optimalno rješenje. Linearni programi minimiziranja mogu također biti neomeđeni ali tada slobodne varijable moraju biti nepozitivne.

Od interesa su nam omeđeni linearni programi. Razlog tome je da simpleks metoda

kreće od jednog rješenja te mu pokušava minimizirati vrijednost funkcije cilja što možemo raditi ako imamo više od jednog rješenja. Ako hoćemo naći optimalno rješenje za omeđeni linearni program morali bi provjeriti vrijednost funkcije cilja za svako rješenje. Broj rješenja u skupu rješenja nam predstavlja problem jer ako se želimo uvjeriti da je rješenje koje smo našli optimalno moramo znati vrijednosti funkcije cilja za sva rješenja.

U tu svrhu uvodimo fundamentalni teorem linearnog programiranja. Teorem tvrdi da ako postoji rješenje za linearni program tada postoji bazično rješenje, te ako postoji optimalno rješenje tada postoji bazično optimalno rješenje. Posljedica ovog teorema je da ako želimo naći optimalno rješenje tada moramo promotriti samo bazična rješenja.

Kako bi dokazali teorem moramo prvo definirati bazično rješenje. Rješenje $X$ linearnog programa je bazično rješenje ako iz rješenja možemo izdvojiti $m$ $x_i$-eva takve da izdvojimo svih $k$ $x_i \neq 0$ i $m - k$ $x_i = 0$ te njihovi pripadajućih stupaca $a_i$ matrice $A$ tvori bazu za $\mathbb{R}^m$. Takve $x_i$-eve zovemo bazičnim varijablama. Bazična rješenja koja imaju manje od $m$ $x_i$-eva koji su veći od nule se zovu degenerativnim bazičnim rješenjima.

Sljedeći dokaz je preuzet iz Luenberger i Ye[2]. Pretpostavimo da je rang matrice $A$ jednak $m$. Kao dokaz prve tvrdnje pretpostavit ćemo da imamo rješenje za linearni program $X$ koje će zadovoljavati jednakost

$$x_1a_1 + x_2a_2 + \cdots + x_na_n = b$$

Pretpostavimo da je imamo $p$ $x_i \neq 0$ te pretpostavljamo da su oni prvih $p$ slobodnih varijabla. Tada znamo da vrijedi

$$x_1a_1 + x_2a_2 + \cdots + x_pa_p = b \tag{2.18}$$

Tada imamo skup koji se sastoji od $a_1, \ldots, a_p$ te on može biti linearno nezavisan ili linearno zavisan.

U slučaju da je nezavisan tada mora vrijediti $p \leq m$. U slučaju da vrijedi $p = m$ rješenje je bazično. U slučaju da je $p$ manji od $m$ možemo naći $m - p$ stupaca iz skupa preostali $n - p$ stupaca takve da je skup od $m$ stupaca linearno nezavisan. Zadajmo vrijednost 0 tim dodanim slobodnim varijablama i dobivamo bazično rješenje.

U slučaju da je zavisan tada imamo ne trivijalnu linearnu kombinaciju tih stupaca koja je jednaka nul-vektoru.

$$y_1a_1 + y_2a_2 + \cdots + y_pa_p = 0$$

Pomnožimo ovu jednakost s proizvoljnim skalarom $\varepsilon$ i oduzmimo je jednakosti 2.18.

$$(x_1\varepsilon y_1)a_1 + \cdots + (x_p - \varepsilon y_p)a_p = b \tag{2.19}$$

Možemo konstruirati vektor $Y$ tako da na prvih $p$ mjesta stavimo vrijednosti $y_i$, te 0 na ostalih $n - p$ vrijednost. Tada imamo funkciju $f(\varepsilon) = X - \varepsilon Y$ koja za svaku vrijednost $\varepsilon$ zadovoljava ograničenja. Povećavamo vrijednost $\varepsilon$ dokad jedna ili više komponenta postanu 0

$$\varepsilon = \{x_i/y_i : \ \ y_i > 0\}$$

Za tu vrijednost rješenje dobiveno iz funkcije ima najviše $p - 1$ pozitivnu varijablu. Ovaj proces možemo ponavljati dokad dođemo do rješenja čiju pripadajući stupci $a_i$ su linearno nezavisni što se svodi na prvi slučaj.

Kao dokaz druge tvrdnje pretpostavimo da imamo optimalno rješenje $X$ i pretpostavimo da imam točno $p$ pozitivnih varijabla $x_1, x_2, \ldots, x_p$. Ponovno imamo dva slučaja jedan gdje su stupci $a_i$ nezavisni i jedan gdje nisu.

Prvi slučaj je isti kao prvu slučaj za prvu tvrdnju ali je nužno dokazati da je novo dobiveno rješenje isto optimalno. Kao dokaz promotrimo vrijednost funkcije cilja za $X - \varepsilon Y$

$$CX - C\varepsilon Y \tag{2.20}$$

Za dovoljno mali $\varepsilon$, rješenje $X - \varepsilon Y$ zadovoljava ograničenja za pozitivni ili negativni $\varepsilon$. Tada zaključujemo da $CY = 0$ jer ako nije onda bi za mali $\varepsilon$ ispravnog predznaka dovela do manje vrijednosti funkcije 2.20 što bi značilo da početni $X$ nije optimalan što je u kontradikciji s pretpostavkom.

Nakon što smo dokazali da je novodobiveno rješenje s manje pozitivnih komponenta optimalno možemo nastaviti dokaz kao u prvom slučaju.

Osim što moramo dokazati tvrdnje fundamentalnog teorema linearnog programiranje moramo dokazati da je broj bazičnih rješenja linearnog programa konačan. Možemo iz stupaca koje smo izdvojili formirati novu matricu $B \in \mathcal{M}_{m\mathsf{X}m}(\mathbb{R})$. Pošto znamo da su stupci matrice $B$ linearno nezavisni tada znamo da postoji jedinstvena linearna kombinacija $X_b$ takva da vrijedi

$$BX_b = b \tag{2.21}$$

Iz ovoga zaključujemo da za svaku kombinaciju $m$ linearno nezavisnih stupaca matrice $A$ postoji točno jedno rješenje $X$ za koje vrijedi

$$AX = b$$

Pretpostavimo da su svaka $m$ različitih stupaca matrice $A$ linearno nezavisni tada moramo odrediti koliko jedinstvenih skupova $m$ stupaca iz $n$ postoji. Konstruiramo skup skupova takvih da oni sadrže $m$ različitih stupaca od $n$ mogućih stupaca te su svi skupovi različiti. Riječ je zapravo kombinaciji bez ponavljanja $n$-tog reda i $m$-tog razreda te je ukupan broj skupova u skupu jednak $\binom{n}{m}$.

Ako želimo odrediti optimalno rješenje za neki omeđeni linearni program,čija je matrica $A$ ranga $m$, morali bi odrediti sve kombinacije $m$ stupaca matrice $A$ koji su linearno nezavisni te riješiti skup $m$ linearnih jednadžbi i $m$ nepoznanica poput 2.21 za svaku kombinaciju.

Ovakav proces za dolaženje do optimalnog rješenje za linearni program je moguć, no ako je razlika $n - m$ velika ukupan broj skupova linearnih jednadžba koji moramo riješiti je ogroman. Ovaj proces je moguće ubrzati.

Recimo da imamo linearni program koji ima bazično rješenje $X$, čije su prve $m$ varijable $x_i > 0$, i želimo pronaći neko rješenje koje je u okolini rješenja $X$ takvo da je vrijednost funkcije cilja za to rješenje manja. Možemo konstruirati vektor $d \in \mathbb{R}^n$ čije slobodne varijable koje nisu povezane s bazom rješenja $X$ su jednake nuli osim jedne koju ćemo zvati $d_j = 1$, te skalara smjera $\rho > 0$. Tada želimo da vrijedi

$$A(X + \rho d) = b \tag{2.22}$$

Iz rješenja $X$ možemo izdvojiti članove baze u $X_b$ i njihove pripadajuće stupce matrice $A$ u matricu $B$ tada vrijedi

$$\begin{gathered} BX_B + \rho Ad = b \\ b + \rho Ad = b \\ \rho Ad = 0 \\ Ad = \sum_{i=1}^{n} a_i d_i = \sum_{i=1}^{m} a_i d_i + \sum_{i=m+1}^{n} a_i d_i \\ 0 = \sum_{i=1}^{m} a_i d_i + a_j \end{gathered}$$

Tada možemo primijetiti da je prva suma umnožak matrice $B$ i članova $d$ povezanih s bazom rješenja $X$ te ćemo te elemente označavati sa $d_B$. Tada vrijedi

$$\begin{aligned} 0 &= Bd_B + a_j \\ d_B &= -B^{-1}a_j \end{aligned} \tag{2.23}$$

Neki bi članovi vektora $d_B$ mogli biti negativni što znači da ako želimo doći do novog rješenja moramo pripaziti na vrijednost od $\rho$ kako ne bi prekršili ograničenja nenegativnosti. Moramo pogotovo pripaziti kod degenerativnih bazičnih rješenja jer ako je $d_i < 0$ i $x_i = 0$ tada bi za takva rješenja najveći $\rho$ koji ne krši ograničenja nenegativnosti bio jednak nuli.

Imamo bazično rješenje $X$, koje na $i$-tom mjestu ima $x_i \neq 0$, možemo odrediti vektor smjera $d$ takav da postoji $\rho > 0$ za koji novo rješenje $X' = X + \rho d$ na $i$-tom mjestu ima $x'_i = 0$ bez da se vrijednost ostalih slobodnih varijabla u rješenje $X$ smanji ispod nule tada imamo bazično rješenje $X' \neq X$ te znamo da se u novoj bazi umjesto stupca $a_i$ nalazi stupac $a_j$. Također je moguće da za nedegenerativni $X$ možemo doći do degenerativnog $X'$ jer može postojati više od jednog $x_i$ čija se vrijednost smanji do $0$ za dani $\rho$.

Kako bi znali kakav utjecaj na vrijednost funkcije cilja ima micanje rješenja $X$ u smjeru $d$ potrebno je promotriti razliku vrijednosti između $X$ i $X'$

$$\begin{aligned} \Delta Z_j &= Z(X') - Z(X) \\ &= C(X + \rho d) - CX \\ &= \rho C d \\ &= \rho(C_B d_B + 1 c_j) \\ \Delta Z_j &= \rho(c_j - C_B B^{-1} a_j) \end{aligned} \tag{2.24}$$

gdje $C_B$ sadrži prvih $m$ cijena. Pošto znamo da je $\rho$ uvijek pozitivan, promjena cijene ovisi o cijeni slobodne varijable koje dodajemo i promjeni vrijednosti slobodnih varijabla u početnom rješenju.

U slučaju linearnog programa dovoljno je da nađemo rješenje za koje ne postoji rješenje s manjom vrijednošću u okolini jer "lokalna optimalnost podrazumijeva globalnu optimalnost"(vlastiti prijevod)[1]. Teorem 3.1 iz Bertsimas i Tsitsiklis[1] tvrdi da ako imamo bazično rješenje $X$ s pripadajućom matricom baze $B$ neka se vektor $\Delta Z$ sastoji od svih $\Delta Z_j$ čiji stupci nisu u bazi. Tada vrijedi

1. Ako $\Delta Z > \mathbf{0}$ tada je $X$ optimalno rješenje
2. Ako je $X$ optimalno i nedegenerativno bazično rješenje tada vrijedi $\Delta Z > \mathbf{0}$

Sljedeći dokazi su preuzeti sa istog izvora[1].

Za dokaz prve tvrdnje pretpostavljamo da vrijedi $\Delta Z > \mathbf{0}$, izaberemo proizvoljno rješenje za $Y$ tada možemo definirati $d = Y - X$. Pošto znamo da rješenja $X$ i $Y$ zadovoljavaju ograničenja znamo da vrijedi $AX = AY = b$ tada mora vrijediti $Ad = \mathbf{0}$. Taj izraz možemo zapisati kao

$$B d_B + \sum_{i \in N} a_i d_i = 0$$

gdje je $N$ skup indeksa koji odgovaraju nebazičnim varijablama. Tada vrijede

$$\begin{aligned} d_B &= -\sum_{i \in N} B^{-1} a_i d_i \\ Cd = C_B d_B + \sum_{i \in N} c_i d_i &= \sum_{i \in N} (c_i - C_B B^{-1} a_i) = \sum_{i \in N} \Delta Z_j d_j \end{aligned}$$

Za svaki nebazični indeks $i \in N$ $x_i = 0$, kako $Y$ zadovoljava ograničenja tada $y_i \geq 0$. Tada $d_i \geq 0$ i $\Delta Z_j d_j \geq 0$ za svaki $i \in N$. Tada $C(Y - X) = Cd \geq 0$ i pošto je $Y$ proizvoljno rješenje, $X$ je optimalno rješenje.

Za dokaz druge tvrdnje pretpostavljanom da je $X$ nedegenerativno bazično rješenje, čijih su prvih $m$ članova različiti od 0, i da je $\Delta Z_j < 0$ za neke $j$. Promotrimo tada promjenu

vrijednosti za kretanje u smjeru bazične varijable $x_i$

$$\Delta Z_i = c_i - C_B B^{-1} a_i$$

Pošto znamo da je $B^{-1}B = \mathbf{I}$ gdje je $\mathbf{I}$ jedinična matrica iz $\mathbb{R}^m$, tada bi umnožak $B^{-1}a_i$ bio jednak vektoru koje na $i$-tom mjestu ima $1$ a na svim ostalima $0$. Tada vrijedi

$$c_i - C_B B^{-1} a_i = c_i - c_i = 0$$

Tada znamo da $j$ mora biti indeks nebazične varijable. Pošto $X$ nije degenerativni, moguće ga je pomaknuti u smjeru vektora $d_j$ te će novodobiveno rješenje imati manju cijenu od $X$ što znači da $X$ nije optimalno rješenje.

Ovaj teorem ima dvije važne posljedice. Kada tražimo optimalno rješenje za linearni program za svako nedegenerativno bazično rješenje možemo odrediti jeli optimalno što znači da ne moramo nužno provjeriti vrijednost funkcije cilja za svako od $\binom{n}{m}$ mogućih bazičnih rješenja, iako ćemo u najgorim slučajevima morati. Također znamo što se događa s vrijednošću funkcije cilja kada pokušavamo neko bazično rješenje $X$ pomaknuti u smjeru $d_j$ tako da novo bazično rješenje $X'$ sadrži bazičnu varijablu $x_j$.

### 2.1.1. Degenerativnost

Kada imamo degenerativno bazično rješenje $X$ za koje svaki $\Delta Z_j$ ima nenegativnu vrijednost tada znamo da je to rješenje, usprkos degenerativnosti, optimalno. U slučaju da imamo degenerativno bazično rješenje $X$ koje ima neki $\Delta Z_j < 0$ s pripadajućim vektorom smjera $d$ čija je komponenta $d_i < 0$ te je pripadajući $x_i = 0$ tada ne možemo lagano zaključiti jeli rješenje $X$ optimalno ili ne.

# 3. Simpleks metoda

George Bernard Dantzig, istaknuti znanstvenik iz ranih dana linearnog programiranja, je nakon Drugog svjetskog rata objavio metodu koja može naći optimalno rješenje za problem s linearnim ovisnostima pod imenom simpleks metoda. Riječ je o metodi za rješavanje linearnih programa koja, osim linearnog programa, uzima jedno bazično rješenje koje iterativnim procesom poboljšava do optimalnog rješenja.

Simpleks metoda ima pravila pomoću kojih možemo izbjeći probleme uzrokovane degenerativnim rješenjima, ali je lakše proći kroz teoriju povezanu sa simpleks metodom da pretpostavimo da su sva bazična rješenja nedegenerativna pa kasnije proširiti teoriju za degenerativna rješenja.

U prošlom odjeljku dokazali smo da ako želimo naći optimalno rješenje za linearni program moramo se samo baviti s bazičnim rješenjima te za svako bazično rješenje znamo je li optimalno, te u slučaju da nije, vrijednost koje slobodne varijable u rješenju trebamo povećati kako bismo smanjili vrijednost funkcije cilja. Što nam nedostaje je proces s kojim možemo s jednog bazičnog rješenja, čija je promjena cijene $\Delta Z_j$ negativna, proći na drugo bazično rješenje, za koje je $x_j > 0$.

Imamo bazično rješenje $X$ čijih su prvih $m$ varijabla bazičnih. Tada vrijedi $AX = b$ što možemo zapisati kao

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1m}x_m + a_{1(m+1)}x_{m+1} + \cdots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{m1}x_1 + \cdots + a_{mm}x_m + a_{m(m+1)}x_{m+1} + \cdots + a_{mn}x_n &= b_m \end{aligned} \tag{3.1}$$

Kako bismo izračunali vektor smjera i promjenu vrijednosti funkcije cilja, za svaku ćemo iteraciju morati izračunati $B^{-1}$. Znamo da vrijediti

$$B^{-1}B = I = \begin{bmatrix} 1 & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & 1 \end{bmatrix}$$

Tada ako zamijenimo neki bazični stupac $a_i$ sa nebazičnim stupcem $a_j$ onda dobivamo novu bazu $B'$ te vrijediti

$$B^{-1}B' = \begin{bmatrix} 1 & \dots & u_1 & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & \dots & u_i & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & \dots & u_m & \dots & 1 \end{bmatrix}$$

gdje stupac koji sadrži $u_k$-eve je povezan sa stupcem $a_j$ koji je dodan u bazu.

Ako $i$-tom retku neke matrice $M$ želimo dodati $j$-oti redak pomnožen sa $k$ možemo je s lijeve pomnožiti matricom $Q = I + kD_{ij}$ ,gdje matrica $D_{ij}$ na svim mjestima ima $0$ osim na mjestu (i, j) koje će biti jednako $1$. Ovakve transformacije zovu se elementarne transformacije nad redcima.

Ako želimo iz matrice $B^{-1}B'$ dobiti jediničnu matricu tada moramo tu matricu pomnožiti s lijeve nekim $Q_u$ koji će $i$-ti redak podijeliti sa $u_i$ te nakon toga dodati svim ostalim redcima $i$-ti redak pomnožen sa $-u_k$ gdje je $k$ indeks retka kojem dodajemo. Tada će $Q_u$ biti umnožak operacije dijeljenja $Q_d$ i svih operacija dodavanja redaka $Q_r$ te će vrijediti $Q_u = Q_m Q_{m-1} \dots Q_{i+1} Q_{i-1} \dots Q_1 Q_d$. Tada će vrijediti

$$\begin{aligned} Q_u B^{-1} B' &= \mathbf{I} \\ Q_u B^{-1} B' B'^{-1} &= \mathbf{I} B'^{-1} \\ Q_u B^{-1} &= B'^{-1} \end{aligned}$$

Na ovaj način možemo održavati matricu $B^{-1}$ kroz iteracije, ali ovo još uvijek zahtjeva da održavamo neku matricu te da radimo množenje s matricama kako bi došli do vektora smjera i vektora promjene vrijednosti funkcije cilja. Takozvana implementacija "puna tablica"(prevedeno iz engleskog "full tableau") se bavi održavanjem simpleks tablice, matrice $B^{-1}[A|b]$, iz koje je trivijalno iščitati promjene vrijednost funkcije cilja i vektore smjera promjene.

Kada promotrimo umnožak $B^{-1}A$ možemo zaključiti iz 2.23 da je riječ o matrici čiji su stupci vektori promjene smjera. Također znamo da vrijedi $BX_B = b$ tada vrijedi $X_B = B^{-1}b$ te će taj stupac sadržavati vrijednosti bazičnih slobodnih varijabla.

Ako odredimo drugu bazu $B'$ tada vrijedi

$$B'^{-1} \begin{bmatrix} A & | & b \end{bmatrix} = Q_u B^{-1} \begin{bmatrix} A & | & b \end{bmatrix}$$

što znači da ako želi početnu simpleks tablicu prebaciti u novu bazu dovoljno je da provedemo elementarne transformacije nad redcima koje tvore matricu $Q_u$. Rezultat ovoga je da eksplicitno pišemo samo prvu matricu $B$, u sve ostale baze prolazimo provođenjem elementarnih transformacija.

Simpleks tablica će također imati redak koji će sadržavati promjenu vrijednosti funkcije cilja te trenutačnu vrijednost funkcije cilja s negativnim predznakom. Početno stanje za ovaj redak u simpleks tablici će biti

$$\begin{aligned} &\begin{bmatrix} \Delta Z_1 & \dots & \Delta Z_n & | & -C_B X_B \end{bmatrix} \\ &= \begin{bmatrix} c_1 - C_B B^{-1} a_1 & \dots & c_n - C_B B^{-1} a_n & | & -C_B X_B \end{bmatrix} \\ &= \begin{bmatrix} c_1 & \dots & c_n & | & 0 \end{bmatrix} + \begin{bmatrix} -C_B B^{-1} a_1 & \dots & -C_B B^{-1} a_n & | & -C_B X_B \end{bmatrix} \\ &= \begin{bmatrix} C & | & 0 \end{bmatrix} - C_B \begin{bmatrix} B^{-1} a_1 & \dots & B^{-1} a_n & | & B^{-1} b \end{bmatrix} \\ &= \begin{bmatrix} C & | & 0 \end{bmatrix} - C_B B^{-1} \begin{bmatrix} a_1 & \dots & a_n & | & b \end{bmatrix} \end{aligned}$$

Bertsimas i Tsitsiklis[1] tvrde da iste elementarne transformacije koje provodimo na drugim redcima simpleks tablice kako bi ih održali u ispravnoj bazi se provode i na ovom retku.

Tada bi ograničenja 3.1 u obliku simpleks tablice izgledala kao

Tablica 1: Simpleks tablica za ograničenja 3.1

| $x_1$ | $\dots$ | $x_m$ | $x_{m+1}$ | $\dots$ | $x_n$ | |
|---|---|---|---|---|---|---|
| 1 | $\dots$ | 0 | $d_{1(m+1)}$ | $\dots$ | $d_{1n}$ | $x_1$ |
| $\vdots$ | | $\vdots$ | $\vdots$ | | $\vdots$ | $\vdots$ |
| 0 | $\dots$ | 1 | $d_{m(m+1)}$ | $\dots$ | $d_{mn}x_n$ | $x_m$ |
| 0 | $\dots$ | 0 | $\Delta Z_{m+1}$ | $\dots$ | $\Delta Z_n$ | $-C_B X_B$ |

Možemo za neku bazu lagano iščitati rješenje

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Znamo da ako postoji $x_j$ takav da je $\Delta Z_j < 0$ tada trenutačno bazično rješenje nije optimalno te ga možemo smanjiti dodavanjem $x_j$ u bazu. Tražimo najveći vrijednost $\rho$ takvu da vrijedi $X + \rho d_j \geq \mathbf{0}$ gdje je $d_j$ stupac smjera promjene za slobodnu varijablu $x_j$. Ovdje možemo primijetiti dva slučaja ovisna o vrijednostima članova vektora $d_j$:

1. Svi članovi vektora $d_j$ su nenegativni u kojem slučaju $\rho$ može biti proizvoljno velik. Iz ovoga možemo zaključiti da je skup rješenja za linearni program beskonačan te da ne postoji optimalno rješenje što zapravo znači da je riječ o neomeđenom programu.

2. Neki članovi vektora $d_j$ su negativni u kojem slučaju $\rho$ poprima konačnu vrijednost koje je jednaka $min\{\frac{x_i}{d_{ji}} : \ d_i < 0 \wedge i \in \mathbb{N}\}$ gdje je $\mathbb{N}$ skup indeksa bazičnih slobodnih varijabla. U ovom slučaju $x_i$ je slobodna varijabla koja napušta bazu.

Provodimo osnovne transformacije na simpleks tablici do kad svaki $d'_{lj} = 0, \ l \in [1, m] \backslash i$ te je $d'_{ij} = 1$. Prvi korak nam je izabrati $j$-ti stupac takvu da $\Delta Z_j < 0 \wedge \exists d_{ij} > 0$. U ovom slučaju $j$ je stupac pivot, dok je $i$ redak pivot, te je $d_{ij}$ element pivot. Tada moramo jednadžbu podijeliti s $d_{ij}$ te svim ostalim jednadžbama oduzeti novonastalu jednadžbu pomnoženu s pripadajućim koeficijentom uz $j$-oti element. Tada dobijemo

Tablica 2: Simpleks tablica sa bazičnim $x_j$ i nebazičnim $x_i$

| $x_1$ | $\dots$ | $x_i$ | $\dots$ | $x_m$ | $x_{m+1}$ | $\dots$ | $x_j$ | $\dots$ | $x_n$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| $1$ | $\dots$ | $d'_{1i}$ | $\dots$ | $0$ | $d'_{1(m+1)}$ | $\dots$ | $0$ | $\dots$ | $d'_{1n}$ | $x_1$ |
| $\vdots$ | | $\vdots$ | | $\vdots$ | $\vdots$ | | $\vdots$ | | $\vdots$ | $\vdots$ |
| $0$ | $\dots$ | $d'_{ii}$ | $\dots$ | $0$ | $d'_{i(m+1)}$ | $\dots$ | $1$ | $\dots$ | $d'_{in}$ | $x_j$ |
| $\vdots$ | | $\vdots$ | | $\vdots$ | $\vdots$ | | $\vdots$ | | $\vdots$ | $\vdots$ |
| $0$ | $\dots$ | $d'_{mi}$ | $\dots$ | $1$ | $d'_{m(m+1)}$ | $\dots$ | $0$ | $\dots$ | $d'_{mn}$ | $x_m$ |
| $0$ | $\dots$ | $\Delta Z_i$ | $\dots$ | $0$ | $\Delta Z_{m+1}$ | $\dots$ | $0$ | $\dots$ | $\Delta Z_n$ | $-C_B X_B$ |

Vidimo da je svaki $d'_{lj} = 0$ osim $d'_{ij} = 1$, također vidimo da $i$-ti element uz sebe ima neke koeficijente. Iz ovog sustava vidimo da je bazično rješenje $X'$ koji ima sve iste bazične slobodne varijable, osim $x_i$ koji je zamijenjen sa $x_j$, jednako

$$X' = \begin{bmatrix} x_1 \\ \vdots \\ 0 \\ \vdots \\ x_m \\ 0 \\ \vdots \\ x_j \\ \vdots \\ 0 \end{bmatrix}$$

Ako ponavljamo ovaj proces dokad više nema $\Delta Z_j < 0$ onda smo stigli do bazičnog optimalnog rješenje, no nije očito da će ovaj proces nakon konačnog broja iteracija doći do optimalnog rješenje ili do zaključka da takvog rješenje nema. Teorem 3.3 iz Bertsimas i Tsitsiklis[1] tvrdi da ako skup rješenja $S$ linearnog programa nije prazan i ako nema bazičnih rješenja koja su degenerativna tada će simpleks metoda u konačnom broju koraka doći do jednog od dva zaključka:

1. Postoji optimalna baza $B$ s pripadajućim bazičnim rješenjem koje je optimalno.
2. Postoji vektor smjera $d_j$ takav da vrijedi $Ad_j = 0,\ d_j \geq \mathbf{0} \wedge \Delta Z_j < 0$ te da je optimalna vrijednost funkcije cilja $-\infty$.

Dokaz za ovaj teorem je također preuzet iz Bertsimas i Tsitsiklis[1].

Ako algoritam završava jer $\Delta Z \geq 0$ onda je uvjeti optimalnosti iz teorema 2.1 zadovoljen te je $B$ optimalna matrica s pripadajućim optimalnim bazičnim rješenjem.

Ako algoritam završava jer za neko bazično rješenje $X$ postoji nebazična slobodna varijabla $x_j$ koja ima $\Delta Z_j < 0$ te su svi članovi vektora promjene $d_j$ nenegativni, tada vrijedi $X + \rho d_j \in S,\ \forall \rho \geq 0$. Tada vrijedi $\forall \rho \geq 0,\ \exists \rho' > \rho$ takav da $Z(X + \rho d_j) > Z(X + \rho' d_j)$ te vrijednost funkcije cilja proizvoljno mala.

Ako bismo ignorirali ovaj smjer i ako svi ostali vektori smjerova imaju negativne članove tada bi simpleks metoda došla do rješenje u konačnom broju koraka te bi tako dobiveno rješenje imalo konačnu vrijednost funkcije cilja koja je nužno manja od proizvoljno mala vrijednosti koje možemo dobiti ako pratimo smjer $d_j$.

## 3.1. Degenerativna rješenja

Izumitelj simpleks metode, George Bernard Dantzig, je na svom osvrtu razvoja linearnog programiranja istaknu da je degeneracije jedna česta u stvarnosti te da su "svi problemi koje je njegova grana Air Forca proučavala ispali degenerativni."[3]

Glavnim problem koji degenerativna rješenje uzrokuju je problem kruženja. Kada imamo simpleks tablicu i bazično degenerativno rješenje tada postoji barem jedan bazični $x_i = 0$. Tada ako izaberemo neki $x_j$ za koji vrijedi da $\Delta Z_j < 0$ kao pivot stupac, te redak koji pripada $x_i$-u kao pivot redak. Tada ako bi proveli osnovne transformacije nad simpleks tablicom nećemo smanjiti vrijednost funkcije cilja u zadnjem retku. Što se zapravo dogodi je zamjena jedne bazične slobodne varijable $x_i = 0$ za drugo $x_j$ koja će također biti jednaka nuli, te su rješenja za linearni program zapravo ista iako im baze nisu.

Ako bi tada $\Delta Z_i$ bio najniži te za izraz $\frac{x_j}{d_{ji}}$ $j$-oti element poprimio minimumu tada ako bi proveli elementarne transformacije nad redcima dobili bi istu simpleks tablicu s kojom smo krenuli.

Ako stignemo do degenerativno rješenje ne znači da ćemo uvijek kružiti te je moguće da ćemo proći više od dvije iteracije prije nego dođemo do početne simpleks tablice.

Za nedegenerativno rješenje $X$ možemo izdvojiti točno jedno $X_B$ i točno jednu matricu $B$ takvu da vrijedi $BX_B = b$. $X_B$ za degenerativno rješenje s pripadajućim $B$ će imati barem jedan član $x_i = 0$ te je moguće taj element zamijeniti nekim nebazičnim elementom $x_j$ ako su $a_i$ i $a_j$ linearno zavisni. Tada bi stupci matrice $B' \neq B$ još uvijek bili linearno nezavisni te bi $X' = X$. Ovo je razlog zašto simpleks metoda može kružiti i zašto teorem 3.3 vrijedi samo za nedegenerativna rješenje.

Za metodu pune tablice ima pravilo koje nam omogućuje da izbjegnemo kruženje zvano "leksikografsko pravilo pivotiranja"(prevedeno iz engleskog "lexicographic pivot rules"). Možemo uspoređivati dva $x, y \in \mathbb{R}^n$ leksikografski. Tada je $x$ leksikografski veći od $y$ ako je prvi element vektora $x - y$, koja nije $0$, pozitivna. Ako vektor $x - y$ nema elementa različitog od $0$ tada je zapravo riječ o istim vektorima. Ako je vektor $x$ leksikografski veći od nul vektora tada je taj vektor leksikografski pozitivan.

Leksikografsko pravilo pivotiranja izmjenjuje kako biramo redak pivot. Umjesto da za proizvoljni stupac $j$ sa $\Delta Z_j < 0$ izaberemo pivot redak povezan sa slobodnom varijablom $x_i$

takav da izraz $\frac{x_i}{d_{ij}}$ poprima najmanju vrijednost za indeks $i$, izabiremo redak pivot povezan sa slobodnom varijablom $x_i$ takav da je $d_{ij} > 0$ te da je taj, ako kao prvi element izaberemo $b_j$, redak podijeljen s $d_{ij}$ leksikografski najmanji.

Teorem 3.4 iz Bertsimas i Tsitsiklis[1] tvrdi da ako imamo simpleks tablicu čiju su redci, osim zadnjeg retka, leksikografski pozitivni tada ako provodimo simpleks metodu pune tablice s promjenama leksikografskog pravila pivotiranja tada vrijedi:

1. Svaki redak, osim zadnjeg, u svakoj iteraciji simpleks metode ostaje leksikografski pozitivan.
2. Zadnji redak se leksikografski isključivo povećava kod svake iteracije.
3. Simpleks metoda završava s konačnim brojem iteracija.

Dokaz za teorem je preuzet iz Bertsimas i Tsitsiklis[1].

Za dokaz prve tvrdnje pretpostavljamo da su svi redci simpleks tablice, osim zadnjeg, leksikografski pozitivni. Neka $j$-ti stupac povezan sa nebazičnom varijablom $x_j$ bude stupac pivot te je $l$-ti redak povezan s $x_l$ redak pivot. Prema leksikografskom pravilu pivotiranja, imamo

$$\frac{(l-ti\ \ redak)}{d_{lj}} < \frac{(i-ti\ \ redak)}{d_{ij}},\ \ ako\ \ i \neq l \wedge d_{ij} > 0$$

U novo simpleks tablici $l$-ti red će biti podijeljen sa $d_{lj}$ prema čemu ostaje leksikografski pozitivan. Pretpostavimo da je $d_{ij} < 0$. Kako bi na tom mjestu dobili $0$ trebamo dodati $l$-ti red pomnožen sa pozitivnim koeficijentom i pošto su oba retka leksikografski pozitivna tada će $i$-ti redak ostati pozitivan. Ako pak pretpostavimo da $d_{ij} > 0$ tada

$$(novi\ \ i-ti\ \ redak) = (stari\ \ i-ti\ \ redak) - \frac{d_{ij}}{d_{lj}}(stari\ \ l-ti\ \ redak)$$

Pošto je $i$-ti redak leksikografski veći tada će novi redak ostati leksikografski pozitivan.

Za dokaz druge tvrdnje znamo da će u pivotom stupcu u zadnjem retku biti broj koji je negativan te ćemo mu morati dodati novi redak $l$ pomnožen s pozitivnim koeficijentom. Pošto je $l$-ti redak leksikografski pozitivan, novi zadnji redak je leksikografski veći od starog zadnjeg retka.

Za dokaz treće tvrdnje znamo da se leksikografska vrijednost zadnjeg retka povećava sa svakom iteracijom tada se nikada ne vraća na prijašnju vrijednost. Pošto vrijednosti zadnjeg retka dobivamo koristeći trenutačnu bazu, tada nijedna baza ne može ponoviti te će simpleks metoda završiti nakon konačnog broja iteracija.

S dodatkom leksikografskog pravila pivotiranja, simpleks metoda pune tablice će uvijek završiti u konačnom broju koraka.

## 3.2. Vodič za provođenje simpleks metode pune tablice

Kako bismo mogli provesti simpleks metodu moramo imati linearni program u standardnoj formi, funkciju cilja $Z : \mathbb{R}^n \to \mathbb{R}$ i ograničenja opisan vektorom $b \in \mathbb{R}^m$ i matricom $A \in \mathcal{M}_{m\mathsf{X}n}(\mathbb{R})$ koja je ranga $m$, i jedno bazično rješenje.

Koraci simpleks metode su sljedeći:

1. Konstruirati matricu $B$ koja se sastoji od bazičnih stupaca matrice A i konstruirati vektore $X_B$ koji se sastoji od vrijednost rješenja bazičnih varijabla i $C_B$ koji se sastoji od cijena bazičnih varijabla.

2. Odrediti inverznu matrice $B^{-1}$.

3. Odrediti umnožak $B^{-1}A$, $B^{-1}b$, $C_BB^{-1}A$ i $C_BX_B$.

4. Konstruirati tablicu oblika
$$\begin{bmatrix} B^{-1}A & | & B^{-1}b \\ - & + & - \\ C_BB^{-1}A & | & -C_BX_B \end{bmatrix} \tag{3.2}$$
Ovaj korak u pravilo nije potreban te je moguće održavati matrice umjesto cijele tablice.

5. Provjerit vrijednosti u zadnjem retku, osim zadnje vrijednosti. Ako među njima nema negativnih brojeva onda metoda završava jer smo stigli do optimalnog rješenja. Ako ima negativnih brojeva tada biramo prvi u čijem stupcu $j$, koji nije zadnji stupac, ima pozitivnih brojeva.

6. Izabrati redak $i$, koji nije zadnji redak i koji u $j$-tom stupcu ima pozitivni broj te je njegov redak podijeljen s vrijednošću od elementa u $j$-tom stupcu i $i$-tom retku leksikografski najmanji.

7. Podijeliti $i$-ti redak sa njegovim $j$-im elementom.

8. Dodati svakom drugom retku, $l$, umnožak $i$-tog retka, $-1$ i $j$-tog elementa retka $l$.

9. Vratiti se na 5. korak

# 4. Programsko rješenje

Prije nego krenemo na programsko rješenje moramo objasniti određene specifičnosti koje će se primjenjivati u programu. Ako znamo da je matrica baza $B$ za neki linearni program jedinična matrica tada vrijedi

$$B^{-1}B = B^{-1}\begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix} = B^{-1}$$
$$B^{-1}B = \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix}$$

što znači da je matrica $B^{-1}$ jedinična matrica. U tom slučaju će simpleks tablica izgledati kao

$$\begin{bmatrix} A & | & b \\ - & + & - \\ C - C_B A & | & -C_B X_B \end{bmatrix} \tag{4.1}$$

U linearni program možemo uvesti nove slobodne varijable na isti način na koji smo ih uveli u 2.8. Tada možemo konstruirati novi linearni program takav da

$$A' = \begin{bmatrix} 1 & \dots & 0 & a_{1(m+1)} & \dots & a_{1n} \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 1 & a_{m(m+1)} & \dots & a_{mn} \end{bmatrix} \tag{4.2}$$

Znamo da će ovako dodane varijable $x_{1'} \dots x_{m'}$ imati stupce $a_{1'} \dots a_{m'}$ koji su linearno nezavisne te će njihove pripadajuće cijene $c_{1'} \dots c_{m'}$ biti jednake $0$. Tada možemo odrediti jedno bazično rješenje čije će bazične varijable biti $x_{1'} \dots x_{m'}$. Tada znamo da vrijedi

$$\begin{gathered} A'X = b \\ B'X_B = b \\ X_B = b \\ \begin{bmatrix} x_{1'} \\ \vdots \\ x_{m'} \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \end{gathered} \tag{4.3}$$

Također znamo da vrijedi

$$Z = C'X = C_B X_B = 0$$
$$C' - C_B A = C' \tag{4.4}$$

Tada će početna simpleks tablica za taj linearni program izgleda kao

$$\begin{bmatrix} A' & | & X_B \\ - & + & - \\ C' & | & 0 \end{bmatrix} \tag{4.5}$$

Matrica $A'$ takvog linearnog programa će imati rang $m$, što je pretpostavka koju smo napravili na početku rada, jer su dodane slobodne varijable linearno nezavisne te će leksikografska vrijednost svakog retka biti pozitiva te će prema teoremu 3.4 linearni program doći do zaključka u konačnom broju iteracija. Ova forma nam također dozvoljava da preskočimo prva tri koraka iz vodiča za provođenje simpleks metode.

Jedan dio programske implementacije će se baviti prikupljanjem linearnih programa, dodavati slobodne varijabli kako bi linearni program zadovoljavao ovo formu i prikazivanjem rješenja za linearni program, dok će drugi dio provjeriti da li linearni program zadovoljava formu i provoditi zadnjih 5 koraka vodiča.

## 4.1. Programska dokumentacija

### 4.1.1. Specifikacija zahtjeva

Programska implementacija simpleks metode, nadalje Programska implementacija, je aplikacija koja bi dozvoljavala korištenje linearnog programiranja za rješavanje problema korisnicima koji nisu upoznati sa linearnim programiranjem. Projektna dokumentacija je namijenjena za vlasnike Programske implementacije čiji će zaposlenici ili kupci koristiti Programsku implementaciju.

Programska implementacije je samostalno rješenje koje će pretvarati opise problema u linearne programe koje će rješavati simpleks metodom te rješenja prikazati korisnicima. Više vrsta korisnika sa različitim problemima će istovremeno koristit aplikaciju, te će se korisnici nalaziti na različitim lokacijama. Također je očekivano da će ukupan broj korisnika i različitih problema koje korisnici susreću rasti s vremenom. Očekivani korisnici bi bili menadžeri koji se često susreću sa kvantitativnim problemima.

Kako bi se Programska implementacija mogla nastaviti razvijati, nužno će biti surađivati sa stručnjacima iz područja za koja želimo proširiti funkcionalnost aplikacije te će se, ovisno o trendovima u tim područjima, postojeća funkcionalnost morat nadograđivat.

Tablica 3: Prvi specifični funkcionalni zahtjev

| | |
|---|---|
| Identifikator | F1 |
| Zahtjev | Programska implementacija će omogućavati rješavanje linearnih programa ispravne forme simpleks metodom |
| Obrazloženje | Korisnici se susreću sa problemima koji se mogu opisati sa linearnim ovisnostima koji su rješivi u konačnom vremenu sa simplex metodom |
| Način provjere | Nakon unosa podataka o problemu, korisniku je ponuđeno optimalno rješenje za upisani problem |
| Prioritet [1-5] | 1 |
| Izvor | Budući korisnici iz različitih područja |

Tablica 4: Drugi specifični funkcionalni zahtjev

| | |
|---|---|
| Identifikator | F2 |
| Zahtjev | Programska implementacija će omogućavati rješavanje više različitih problem |
| Obrazloženje | Korisnici programske implementacije će biti iz različitih područja te će problemi s kojima se susreću biti različiti |
| Način provjere | Korisnik može izabrati obrazac koji odgovara njegovom problemu |
| Prioritet [1-5] | 2 |
| Izvor | Budući korisnici iz različitih područja |

#### 4.1.1.1. Dinamika realizacije zahtjeva

Inicijalna verzija će u potpunosti realizirati samo prvi zahtjev:

- F1 - Programska implementacija će sadržavati potpunu i testiranu implementaciju simpleks metode za rješavanje linearnih programa
- F2 - Programska implementacija će imati implementirano dva obrazac za dva različita problema i dovršeni okvir za dodavanje novih obrazaca

Naredna verzija će u potpunosti realizirati drugi funkcionalni zahtjev:

- F2 - Programska implementacija će imati 10 implementiran obrazaca za rješavanje različitih problema

#### 4.1.1.2. Nefunkcionalni zahtjevi

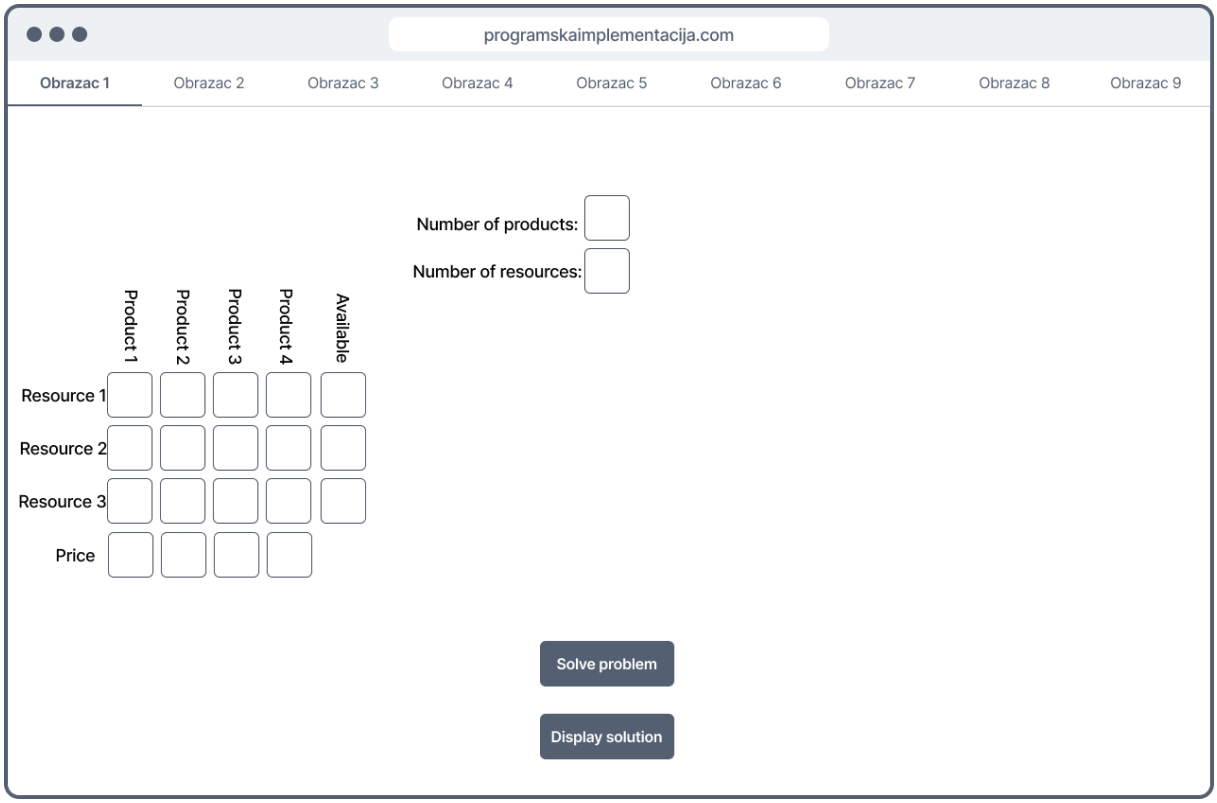
- Izgled
  - NF1 - Programska implementacija će interakciju s korisnikom provoditi putem grafičkog sučelja
- Performanse

  - NF2 - Programska implementacija će za upisani problem doći do zaključka u prosjeku manje od 10 sekundi

- Izvođenje softvera i okruženje
  - NF3 - Programska implementacija će biti dostupna korisnicima kroz web-preglednik

#### 4.1.1.3. Skica

Na slici 5 je prikazana skica Programske implementacije. Pri vrhu će imati meni na koje će se moći izabrati obrazac. Obrazac koji je prikazan je zamišljen za rješavanje problema proizvodnje te bi budući obrasci mogli izgledati bitno drugačije.



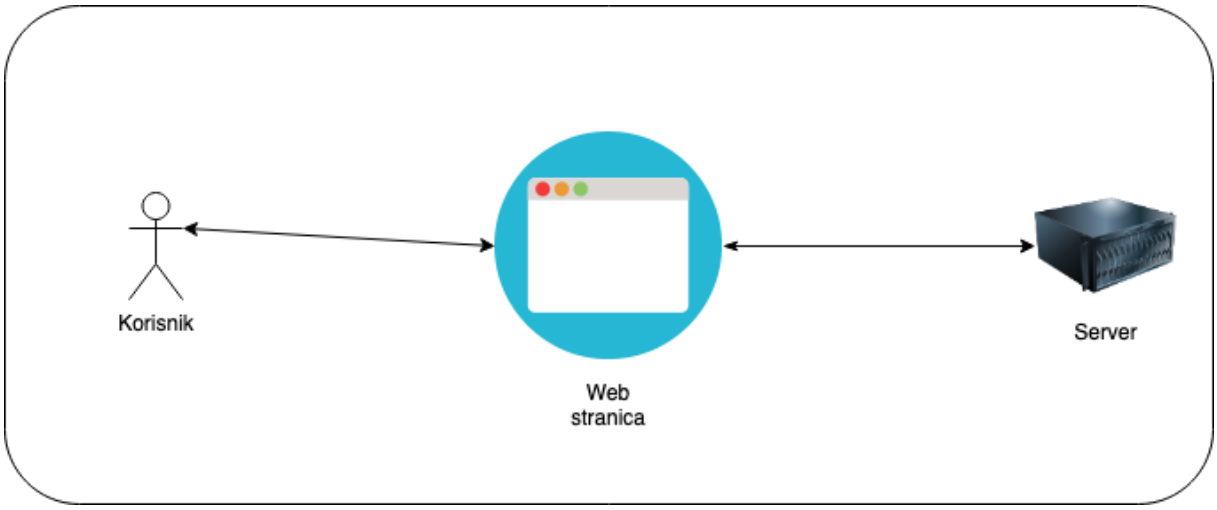

Slika 5: Konceptualni prikaz arhitekture programskog rješenja

### 4.1.2. Konceptualni model

Kako bi ispunili funkcionalni zahtjev, Programska implementacija će biti realizirana u obliku web aplikacije što će ujedinio i ispuniti drugi nefunkcionalni zahtjev. Web stranica će biti napisana u Reactu dok će server biti napisan u Rustu.

Ponašanje sustava opisat ćemo dijagramom aktivnosti.

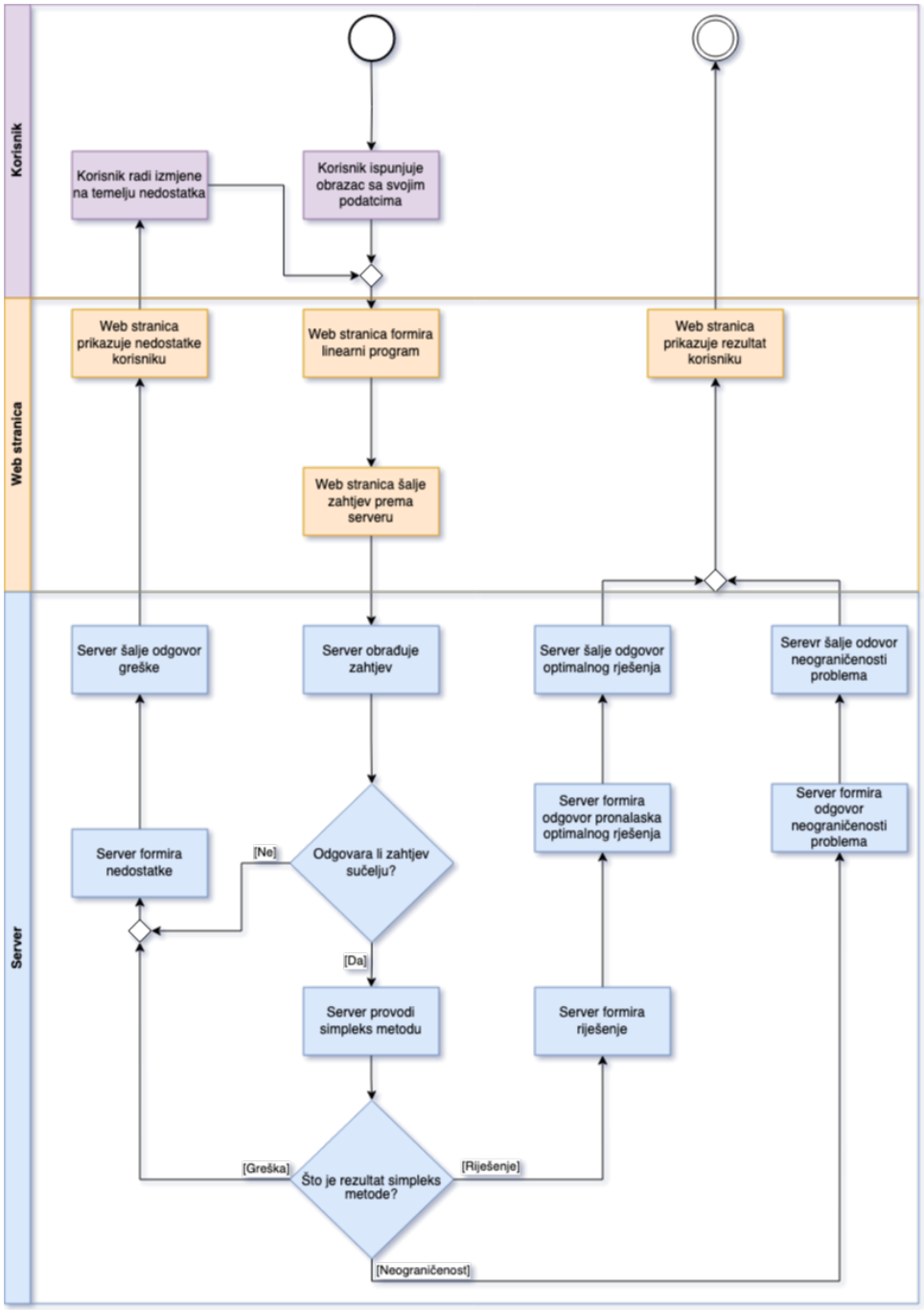
Korisnici će se prijavljivati na web stranicu u kojoj će, kroz obrasce koji su izrađeni za jednu namjenu, rješavati svoje probleme. Web stranica će primati podatke za linearni program od korisnika te će oblikovati i poslati zahtjev prema serveru. Server će primiti podatke o linearnom programu i izvršiti provjere forme primljenog linearnog programa.




Slika 6: Konceptualni prikaz arhitekture programskog rješenja

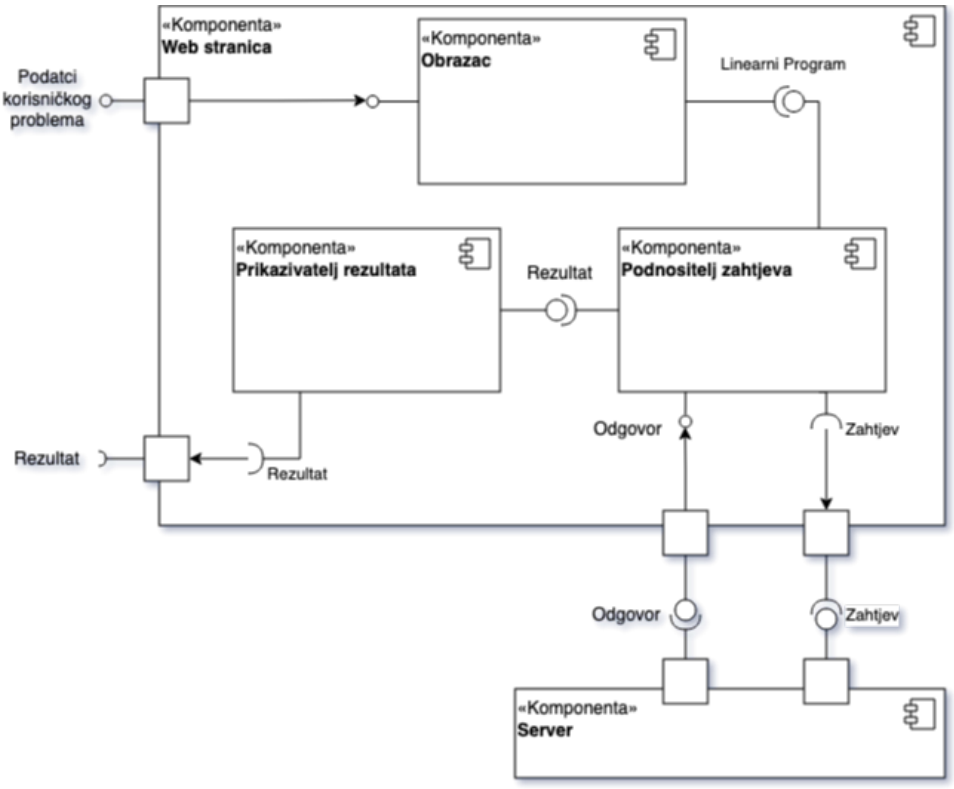
U slučaju da linearni program ne zadovoljava formu, server formira odgovor greške koji sadrži podatke o nedostatcima u poslanom programu te ih prosljeđuje web stranici koja prikazuje relevantne informacije korisniku.

U slučaju da linearni program zadovoljava formu, server počne provoditi simpleks metodu koja ima tri moguća rezultata. Prvi rezultat je pronađeno optimalno rješenje u kojem slučaju server formira odgovor koji sadrži optimalno rješenje i optimalnu vrijednost funkcije cilja koju šalje web stranici. Web stranica korisniku javlja da je optimalno rješenje pronađeno te mu prikazuje optimalno rješenje i vrijednost funkcije cilja. Drugi rezultat je određena neograničenost problema, tada server formira odgovor o neograničenosti problema i šalje ga web stranici. Web stranica korisniku javlja da je njihov problem neograničen te ovisno o problemu javlja da je optimalni trošak $-\infty$ ili $\infty$. Treća mogućnost je da dođe do greške tijekom provođenja simpleks metode u kojem slučaju server formira odgovor greške i šalje ga web stranici na isti naći kao kada primljeni program ne zadovoljava formu.




Slika 7: Konceptualni prikaz procesa programskog rješenja

Strukturu programske implementacije ćemo prikazati s dijagramom komponenta.




Slika 8: Dijagram komponenta programskog rješenja

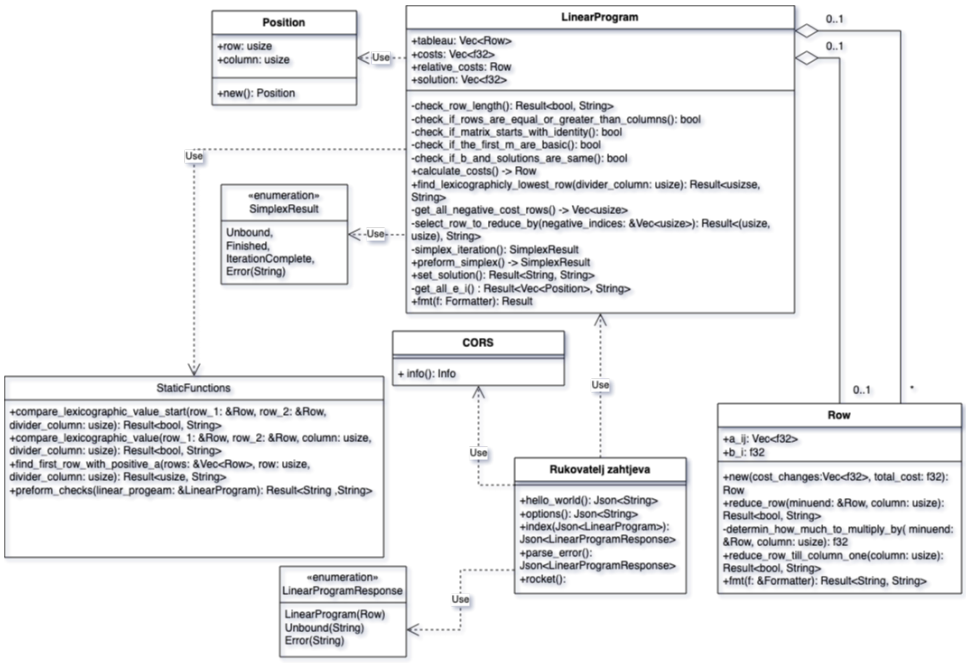
Na njemu vidimo prikaze dvije komponente, Web stranica i Server, gdje Web stranica u sebi sadrži druge komponente. Web stranica komunicira s korisnikom preko Obrazaca koji upisane podatke transformira u Simpleks tablicu. Simpleks tablica je nakon toga poslana Podnositelju zahtjeva koji pakira Simpleks tablicu i podnosi Zahtjev prema Serveru. Kada primi Odgovor otpakira ga te ga predaje Prikazivatelju rezultat koji Rezultate na zaslonu pokazuje korisniku.

U pravilo bilo koja od komponenata se može zamijeniti ali je u sklopu ove programske implementacije bilo jedino važno da Obrazac bude komponenta kako bi se obrasci mogli izmjenjivati po potrebi. Podnositelj zahtjeva i prikazivatelju rezultat nisu morali biti komponente, ali ih je u Reactu bilo najlakše izgraditi kao komponente.

Server prima zahtjev te ga obrađuje nakon čega ga vraća Web stranici.

### 4.1.3. Fizički modeli

Na fizičkoj razini ćemo promotriti web stranicu i server razdvojeno jer se komunikacija među njima odvija isključivo preko HTTP zahtjeva te obje komponente za tu funkcionalnost imaju točno jednu funkciju.




Slika 9: Dijagram klasa za Server komponentu

Iako Rust nema klase moguće je nad strukturama implementirati funkcije koje primaju samu strukturu kao ulaz što je dovoljno blizu klasama da možemo strukturu Servera prikazati dijagramom klasa. Tipični dijagrami klasa bi imali veći broj kompozicija ali pošto Rust nije objektno orijentirani jezik toga ovdje nema u tolikoj mjeri. `LinearniProgram` je centralna klasa za Server komponentu. U njoj se nalaze svi podatci potrebni za odvijanje simpleks metode uključujući ograničenja pod imenom `tableau`, cijene slobodnih varijabli pod imenom `costs`, promjene vrijednosti funkcije cilja i vrijednost funkcije cilja po imenom `relative_costs` te rješenje pod imenom `solution`. Ona sadrži 5 privatnih metoda koje provjeravaju odgovara li problem traženoj formi te se pozivaju pomoću metode `preform_checks` iz `StaticFunctions` klase. Za obavljanje leksikografskih uspoređivanja postoji metoda `find_lexicographicly_lowest_row` te metode `compare_lexicographic_value_start`, `compare_lexicographic_value` i `find_first_row_with_positive_a` na klasi `StaticFunctions`. Ostale metode služe za obavljanje simpleks metode.

Klasa `Row` opisuje jedno ograničenje sa svojim koeficijentima uz slobodne varijable i konstantom na desnoj strani ograničenja. Ona, osim konstruktora, ima metode za obavljanje elementarnih transformacija koje `LinearProgram` pozivaju tijekom odvijanja simpleks metode.

Uz njih su povezane klasa `Position`, koja je pomoćna klasa te se koristi u metodi `get_all _e_i` iz klase `LinearProgram` za dohvaćanje svih bazičnih slobodnih varijabli, i

enumeracija SimplexResult koja služi za izvještavanje statusa simpleks metode.

`Rukovatelj zahtjeva` je klasa napisana u Rocket frameworku koja obavlja sva primanja zahtjeva i odgovaranja na zahtjeve. Metoda `rocket` postavlja rute za metode `hello_world`, `options` i `indeks` kao metode koje odgovaraju na HTTP zahtjeve, one također provode konverziju iz json string u klasu i iz klase u json string. Enumeracija `LinearProgramResponse` diktira strukturu podatak koji će biti poslani kao odgovor, dok je klasa `CORS` pomoćna klasa koja diktira s čim Server može komunicirati i kako može komunicirati.

Kad Server primi HTTP zahtjev, koji ima rutu i metodu, funkcija `rocket` određuje kojoj funkciji proslijediti zahtjev. U slučaju da ruta nije "/" odgovor koji Server vraća je standardna error poruka tip 404. U slučaju da primi GET zahtjev vraća se odgovor koji u tijelu sadrži "Hello world". Ako primi OPTION zahtjev šalje podatke o postavkama Cross-origin resource sharinga. U slučaju da dobije POST zahtjev tijelo zahtjeva se pokuša deserializirati u obliku `LinearProgram`, u slučaju da se ne može odgovor koji server vraća je standardni error tip 422.

Ako se tijelo može deserializirati onda se `LinearniProgram` predaje `indeks` funkciji `Rukovatelja zahtjeva`. indeks funkcija prvo poziva `preform_checks` funkciju iz `StaticFunctions` koja poziva sve funkcije na `LinearProgramu` koje provjeravaju pojedine aspekte forme `LinearPrograma`. U slučaju da ovdje dođe do greške ili se otkrije nedostatak u formi `LinearnogPrograma`, onda `indeks` funkcija vraća odgovor u čijem se tijelu nalazi podatka o grešci koja se dogodila ili nedostatku u formi. Ako `LinearProgram` prođe sve provjere onda se na njemu poziva metoda `calculate_costs` koja postavlja početne promjene vrijednosti funkcije cilja za sve stupce. Nakon toga se na `LinearProgramu` poziva `preform_simplex` metoda koja na `LinearProgramu` poziva `simplex_iteration` dokad ono rezultira s `Finished`, `Unbound` ili `Error` iz enumeracije `SimplexResult`.

Metoda `simplex_iteration` pronalazi stupac s najnižim indeksom koji ima negativnu promjenu vrijednosti funkcije cilja i ima redak koji u tom stupcu ima broj veći od nule. Ako nema stupca s negativnom promjenom vrijednosti onda metoda vraća poruku završenja simpleks postupak što znači da je optimalno rješenje nađeno. Ako Postoje stupci s negativnom promjenom vrijednosti, tada se poziva metoda `select_row_to_reduce_by` koja prolazi po stupcima i poziva metodu `find_lexicographically_lowest_row`.

Ta metoda prvo pokušava naći jedan redak koji ima u tom stupcu broj veći od 0, ako takvog retka nema onda vraća Err te metoda `select_row_to_reduce_by` prelazi na sljedeći stupac. Ako takvog retka ima onda pokušava naći sljedeći takav redak. Ako drugog retka nema onda vraća indeks prvog retka.

Ako takvog retka ima onda se pomoću funkcije `compare_lexicographic_value_start` redci uspoređuju leksikografski. Izabere se onaj manji te se pokušava naći sljedeći redak koji ima u stupcu broj veći od 0 te se proces ponavlja. Ako takvih redaka više nema onda se vraća najmanji pronađeni redak.

Ako metoda `select_row_to_reduce_by` ne nađe odgovarajući stupac onda se poruka koja nam govori da je program neograničen. Ako se stupac i redak nađu onda metoda

`select_row_to_reduce_by` vraća indeks tog stupca i retka.

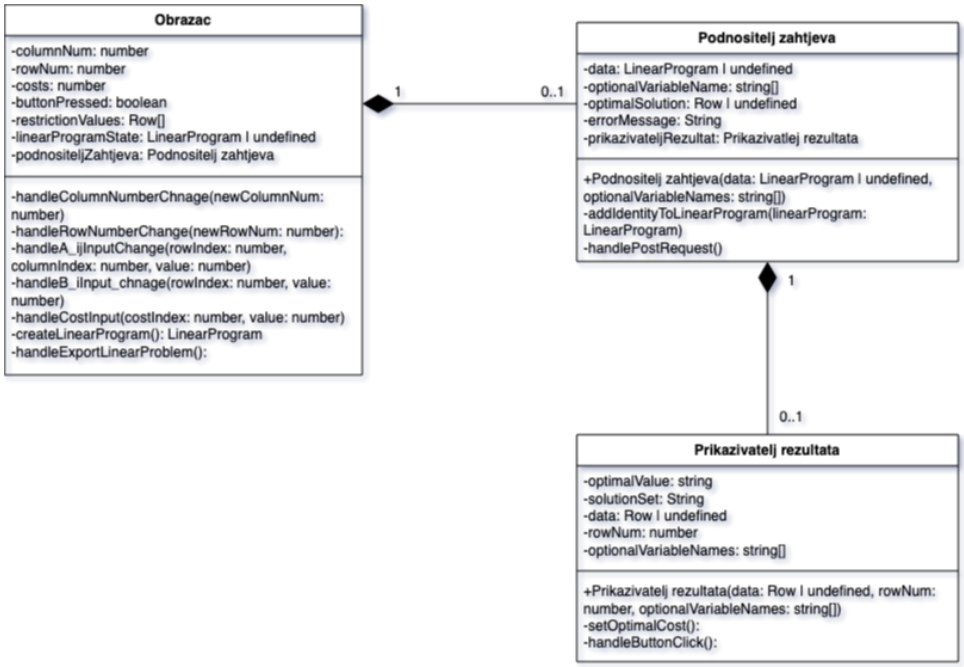
Nakon toga pozivamo metodu `reduce_row_till_column_one` nad retkom i indeksom stupca koji je izabran te se vrijednosti u cijelom retku dijele sa elementom koji se čiji je indeks jednak indeksu stupca.

Nakon toga se nad svim ostalim redcima poziva `metoda reduce_row` s izabranim retkom i indeksom stupca koji dodaje svojem retku izabrani redak pomnožen s brojem koji je vraćen pozivanjem metode `determine_how_much_to_multiply_by` nad tim retkom. Isti se proces provede nad retkom relative_costs LinearPrograma. Nakon toga metoda `simplex_iteration` vraća poruku uspješnosti provođenja simpleks iteracije.

Ovaj proces se ponavlja se dokad metoda `simplex_iteration` ne vrati poruku uspješnosti provođenja simpleks iteracije.

Ako metoda `preform_simplex` rezultira porukom neograničenosti ili greškom, tada funkcija `indeks` vraća odgovor koji u tijelu sardži odgovarajuću obavijest korisniku.

Ako metoda `preform_simplex` rezultira s porukom završenja simpleks postupka onda se nad `LinearProgramom poziva` metoda `set_solution` koja za sve indekse stupca koje metoda `get_all_e_i` dohvati postavlja vrijednosti polja solution na `LinearProgramu`. Nakon toga se vrijednost tijelo odgovora postavlja na rješenje koje se nalazi na polju solution te se odgovor šalje Web stranici.




Slika 10: Dijagram klasa za Web stranicu

Pošto je Web stranica napisana u Reactu ovdje nije riječ o klasam već React komponentama, no dovoljno su bliske klasama da se mogu prikazati na dijagramu klasa.

Klasa `Obrazac` je klasa koja sadrži najmanji broj potrebnih atributa i metoda kako bi mogla funkcionirati. U UML ovo bi se prikazivalo preko nasljeđivanja ali React nema koncept za nasljeđivanje u komponentama. Kako korisnik popunjuje polja na zaslonu tako se pozivaju metode

koje te podatke upisuju u polja. Na klik gumba se pokrene metoda `handleExportLinearProblem` koja poziva metodu `createLinearProgram`, koja iz upisanih podatak stvara objekt sličan klasi `LinearProgram` iz Server komponente, i isključuje isti gumb dokad korisnik izmjeni podatke.

Kada se pripremi objekt kreira se `Podnositelj zahtjeva`. Na klik gumba za slanje pokrene se metoda `handlePostRequest` koja prvo poziva metodu `addIdentityToLinearProgram`. Ta metoda izmjenjuje objekt tako da zadovoljava formu 4.5. Nakon modifikacije `handlePostRequest` podnosi zahtjev na server i čeka odgovor.

Ovisno o dobivenim podatcima postavlja vrijednost `errorMessage` koja se prikazuje na zaslonu. Ako je za objekt nađeno rješenje onda se kreira `Prikazivatelj rezultata` koji na klik gumba izvršava `handleButtonClick` koji poziva `setOptimalCost` koji postavlja dobiveno rješenje na zaslon.

### 4.1.4. Razvoj programskog rješenja

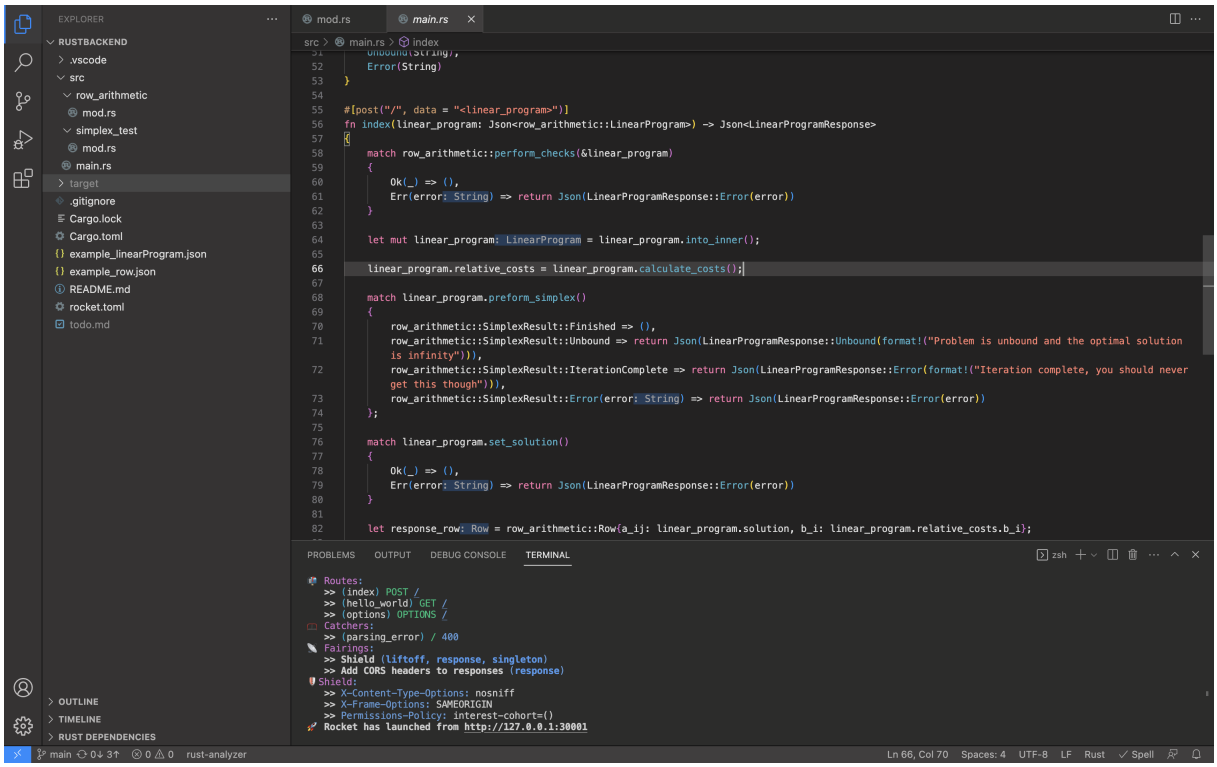
Za razvoj programske implementacije, prvo se krenulo u razvoj implementacije simpleks metode u kodu. Rust dolazi sa programom Cargo koji se koristi za upravljanje Rust projekta te je prvo sa Cargo kreirana inicijalna verzija projekta. Rust dozvoljava da kod podijelimo u module te smo sav kod koji ima veze sa provođenjem simpleks metode staviti u jedno mjesto. Moguće je provesti jedinične testove na odvojenom modulu te su se nakon razvoja simpleks metode napisali i proveli jedinični testovi.

Nakon razvoja simpleks metode u projekt je dodan crate Serde koji dozvoljava serializiranje i deserializiranje json teksta u obliku podataka. Kako bi se implementirala i testirala ta funkcionalnost dodane su dvije privremene metode na klasu `LinearProgram` koje obavljaju tu funkciju, te je u program dodan kod koji čita iz lokalnog file i stvara novi koji sadrži json oblik riješenog `LinearProgram`

Kada je ta funkcionalnost osposobljena dodan je crate Rocket, web razvojni okvir koji Programskoj implementaciju pretvara u web server te ima integraciju sa Serde pa se funkcionalnost prije navedenih privremenih metoda prebacila na Rocket. Kreirane su metode za obradu GET, POST i OPTION HTTP zahtjeva. Funkcija `index` je stvorena kao funkcija odgovorna za POST zahtjeve te je u nju stavljen kod koji poziva modul koji implementira simpleks metodu. Web server je prvo testiran sa aplikacijom Postman nakon čega se krenulo u razvoj Web stranice.

Za razvoj Server komponente koristio se uređivač koda Visual Studio Code, što se može vidjeti na slici 11, sa ekstenzijama rust-analyzer, Rust Syntax, Even Better TOML, crates i CodeLLDB. Na slici se također vidi struktura programske mape. Sav kod se nalazi u "src" mapi. Kod klasa `Rukovatelj zahtjeva` i `CORS` i enumeracije `LinearProgramResponse` se nalazi u "src/main.rs", dok se kod klasa `LinearProgram`, `Row`, `Position` i `StaticFunctions` i enumeracije `SimplexResult` nalazi u datoteci "src/row_arithmetic/mod.rs". Kod jediničnih testova se nalazi u datoteci "src/simplex_test/mod.rs". Mapa "target" je mapa koja sadrži kompilirane datoteke. Još se mogu istaknuti "Cargo.toml" i "rocket.toml" koje su konfiguracijske

datoteke projekta, te datoteka "Cargo.lock" koja je datoteka koja sadrži podatke o ovisnostima korištenih crateova te je automatski generirana pri kompilaciji programa. Još su prisutne datoteke za git repozitorij poput ".gitignore" i "README.md" i mape ".git", te mapa ".vscode" koja sadrži lokalne konfiguracije za ekstenzije i datoteka "todo.md" koja sadrži stvari koje se još trebaju obaviti.

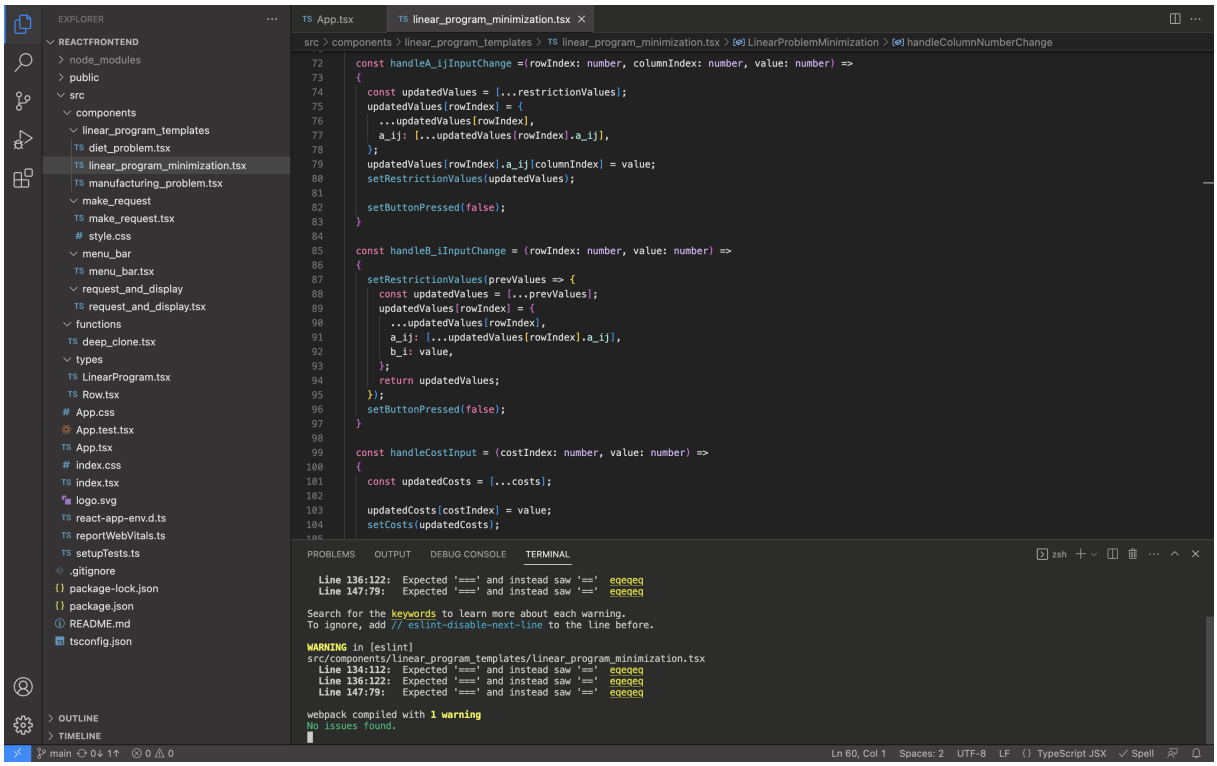



Slika 11: Razvojno okruženje za Server komponentu

Za kreaciju i pokretanje web stranice se koristio Node program. Web stranice je kreirana naredbom za inicijaliziranje React projekta sa TypeScript obrascem. Prvo se razvila komponenta Podnositelj zahtjeva kako bi se mogla testirati interakcija Web stranice i Servera. Server u ovom trenutku nije imao CORS podatke te su oni dodani i Web stranica je uspjela podnijeti zahtjev prema Serveru. Nakon toga je razvijeni prvi Obrazac kako bi mogli na Web stranici formirati podatke i poslati ih na Server. Kada se Obrazac završio krenulo se u razvoj komponente Prikazivatelj rezultata.

Na slici 12 prikazano je programsko okruženje za razvoj komponente Web stranica. Ponovno je riječ o uređivaču koda Visual Studio Code, ali u ovom slučaju nema ekstenzija. Na lijevoj stani se vidi struktura datoteka i ona malo kompleksnija od strukture Server komponente. Naredba koja inicijalizira program stvori više datoteka za početak te su samo mapa "src/components/" i njezini sadržaji kreirani tijekom pisanja rada. Sve ostale datoteke, osim "src/App.tsx", nisu izmijenjene. Datoteka "src/App.tsx" ke ekvivalent main datoteke u drugim programskim jezicima te ona sadrži samo sadrži `MenuBar` komponentu čiji se kod nalazi u datoteci "/src/components/menu_bar/menu_bar.tsx". Programski kod komponente Podnositelj zahtjeva se nalazi u "src/components/make_request/make_request.tsx", dok se programski kod Prikazivatlja rezultata nalazi na "src/components/request_and_display/request_and_display.tsx". Programski

kod obrazaca se nalazi u mapi "src/components/linear_program_templates" gdje je svaka datoteka u toj mapi jedan obrazac.




Slika 12: Razvojno okruženje za Web stranica komponentu

### 4.1.5. Testiranje programskog rješenja

#### 4.1.5.1. Ciljevi testiranja

Glavni ciljevi procesa testiranja su:

- Potvrditi ispravnost provođenja simpleks metode.
- Potvrditi ispravnost provođenja provjera forme linearnog programa.
- Osigurati da se Server neće srušiti zbog ulaznih podataka.

#### 4.1.5.2. Vrste testiranja

Proces testiranja će sadržavati sljedeće vrste testiranja:

1. **Jedinično testiranje:** Testiranje logike Server komponente za obradu zahtjeva
2. **End-to-end testiranje:** Testiranje cijele Programske implementacije na način na koji bi korisnici koristili

#### 4.1.5.3. Jedinično testiranje

Kada Server dobije zahtjev podatci u tijelu se pokušavaju pretvoriti u `LinearProgram` što znači da trebaju zadovoljavati strukturu propisanu na dijagramu klasa. Taj proces zapravo provodi jedan dio provjere podataka te nam garantira da će proslijeđeni podatci barem zadovoljavati tu strukturu. Provjere podataka u strukturi moramo provoditi sami te su u tu svrhu napisane metode. Rust ima alate koji dozvoljavaju jedinično testiranje u obliku modula. Svaki testni scenarij će pripremiti `LinearProgram` koji će imati grešku koju provjere trebaju uhvatit te će se nakon provođenja provjeriti vraćena vrijednost. Testni scenariji su sljedeći:

- Testni scenarij 1: Pripremi se `LinearProgram` takav da jedan `Row` u `tableau` ima manje elementa u `a_ij` od prvog `Row`. Program treba primijetiti nedostatak i vratiti grešku sa porukom o nejednakosti brojeva elementa.
- Testni scenarij 2: Pripremi se `LinearProgram` takav da je ukupan broj `Row` u vektoru `tableau` veći od brojeva elementa u `a_ij` u prvom `Row`. Program treba primijetiti višak i javiti grešku sa porukom o višku retka u linearnom programu.
- Testni scenarij 3: Pripremi se `LinearProgram` takav da `tableau` kada se promotri u obliku matrice ne počinje sa jediničnom matricom. Program mora primijetiti nedostatak i vratiti grešku sa porukom o neispravnosti početka matrice ograničenja linearnog programa.
- Testni scenarij 4: Pripremi se `LinearProgram` takav da mu vrijednosti na `solution` imaju negativne brojeve. Program mora primijetiti da na `solution` postoje negativne brojevi i javiti grešku s porukom da rješenje sa negativnim brojevima nije moguće.

- Testni scenarij 5: Pripremi se `LinearProgram` takav da mu `solution` na mjestima s indeksom većim od broja `Row` na `tableau` ima vrijednosti različite od 0. Program treba primijetiti da su te vrijednosti veće od 0 i javiti grešku sa porukom o tome da samo mjesta s indeksom manjim ili jednakim broju `Row` na `tableau` mogu imati vrijednost veću od 0.
- Testni scenarij 6: Pripremi se `LinearProgram` takav da elementi `solution` nisu jednaki elementima na `b_i` na `Row` na `tableau`. Program mora primijetiti neusklađenost i vratiti grešku sa porukom o neusklađenosti.
- Testni scenarij 7: Pripremi se `LinearProgram` takav da zadovoljava formu 4.5 i da je neograničen. Program treba vratiti odgovor neograničenosti.
- Testni scenarij 8: Pripremi se `LinearProgram` takav da zadovoljava formu 4.5 i da ima optimalno rješenje. Program mora vratiti uspješno sa porukom koja sadrži `Row` čija su polja `a_ij` postavljena na vrijednosti optimalnog riješena i polje `b_i` vrijednost funkcije cilja za to rješenje

#### 4.1.5.4. End-to-end testiranje

Sa end-to-end testiranjem provjeravamo međudjelovanje komponenata, transformaciju ulaznih podataka u objekt sličan objektima `LinearProgram` i prikaz dobivenih odgovora od servera. Također provjeravamo što se dogodi kada serveru proslijedimo podatke koji nisu brojke. Testni scenariji su sljedeći:

- Testni scenarij 1: U obrazac upisujemo podatke takve da bi linearni program izgrađen iz njih imao optimalno rješenje te prolazimo kroz korake za pošiljanje zahtjeva i prikazivanje rezultata. Očekujemo da će se na zaslonu pojaviti poruka o uspješnom rješavanju linearnog programa, prikaz optimalne vrijednosti funkcije cilja i optimalnog rješenja.
- Testni scenarij 2: U obrazac upisujemo podatke takve da bi linearni program izgrađen iz njih bio neograničen te prolazimo kroz korake za pošiljanje zahtjeva. Očekujemo da će se na zaslonu pojaviti vijest o neograničenosti programa.
- Testni scenarij 3: U obrazac upisujemo podatke takve da je barem jedno polje nema vrijednost te prolazimo kroz korake za pošiljanje zahtjeva i prikazivanje rezultata. Očekujemo da će se na zaslonu pojaviti poruka o neočekivanom simbolu, ali da se ni Web stranica ni Server neće srušiti.
- Testni scenarij 4: U obrazac upisujemo podatke takve da je barem jedno polje ima u sebi vrijednost koja nije brojka (u našem slučaju string sa samo alfabetskim simbolima) te prolazimo kroz korake za pošiljanje zahtjeva i prikazivanje rezultata. Očekujemo da će se na zaslonu pojaviti poruka o neočekivanom simbolu, ali da se ni Web stranica ni Server neće srušiti.

#### 4.1.5.5. Rezultati testiranja

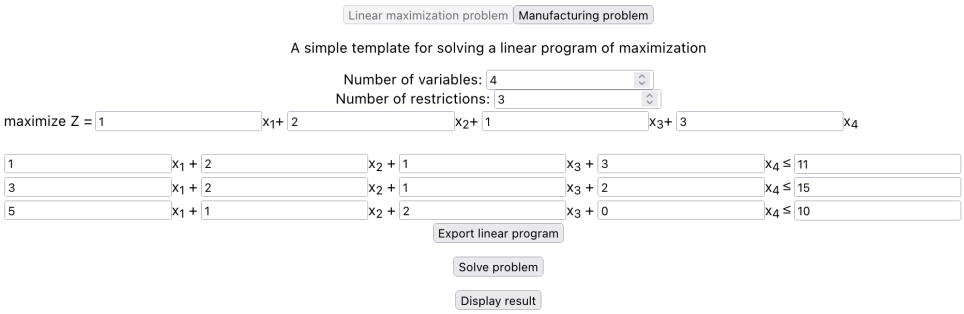
Kada smo prvi put provodili jedinični test uočili smo neočekivani rezultat u testnom scenariju 5 i 7. Testni slučaj 5 nije vraćao grešku već je pokušao provesti simpleks metodu na `LinearProgram` te je rješenje koje je dobilo bilo neispravno. Greška se nalazila u metodi `check_if_the_first_m_are_basic` koja je provjeravala koji je ukupan broj elementa veći od nula na polju `solution`. Greška je ispravljena tako da metoda provjerava postoji li element s indeksom većim broja `Row` na polju `tableau`. Testni slučaj 7 je umjesto odgovor neograničenosti slao odgovor greške sa porukom koja je tvrdila da je program neograničen. Greška je ispravljena tako da je odgovor koji implementacija vraća zamijenjen za odgovor neograničenosti.

Nakon prvog provođenja jedinični testovi više nisu pali.

Kada smo prvi put proveli end-to-end testiranje testni scenariji 1, 2 i 3 su prošli kako smo očekivali. Testni scenariji 4 je nakon upisa u polje za broj ograničenja dogodila se greška na Web stranici. Kada vrijednost tog polja postavimo na nešto što nije broj, broj stupaca se redaka se pokušava postaviti na nešto što nije broj te se dogodi greška koja zahtjeva da se korisnik ponovno spoji na Web stranicu. Greška je uklonjena dodavanjem provjere upisane vrijednosti te postavljanjem vrijednosti na 2 ako ona nije brojka.

## 4.2. Korisničke upute

Kako bi korisnici htjeli koristi programsko rješenje prvo se moraju spojiti na web stranicu putem linka. Kada se spoje će vidjeti sljedeće na zaslonu




Slika 13: Izgled web stranica kada se prvi puta na nju dođe

Inicijalni obrazac je obrazac linearnog programa za pronalaženje maksimuma funkcije cilja s ograničenjima manje ili jednako. Ovaj obraza pretpostavlja da su slobodne varijable nenegativne.

Korisnik može namjestiti broj slobodnih varijabla i ograničenja preko dviju polja uz labele "number of varialbes" i "number of restrictions".

Vrijednost za oba polja ne može biti manja od 2, te broj ograničenja ne može biti veći od broja slobodnih varijabla.

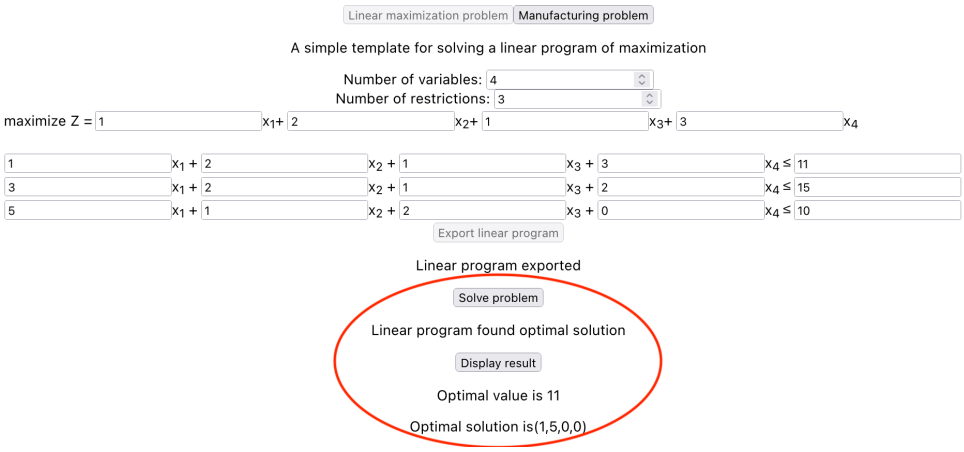
Ispod tih polja se nalaze polja za cijene u funkciji cilja koja se upisuju uz pripadajuću slobodnu varijablu.

Vrijednosti u tim poljima mogu biti bilo koji brojevi iz $\mathbb{R}$. U ovom slučaju imam 4 slobodnih varijabli pa je i toliko polja u funkciji cilja.

Ispod toga se nalaze polja za ograničenja. Svaki redak je jedno ograničenje, dok su polja koeficijenti uz slobodne varijable i konstante na lijevoj strani nejednakosti.

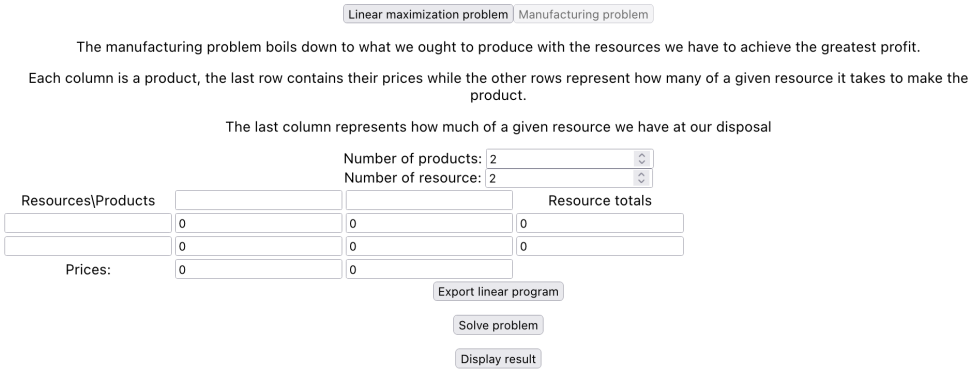
Ovaj konkretni linearni program će imati 4 slobodne varijable i 3 ograničenja. Kada su svi podatci upisano korisnik mora kliknuti gumb na koje piše "Export linear program". Na zaslonu bi se trebala pojaviti poruka kada se zadatak obavi te se gumb više ne može stisnuti. Ako korisnik želi napraviti izmjene na linearnom programu može bilo koje polje promjeriti nakon čega će se gumb ponovno moći stisnuti.

Nakon toga korisnik može kliknuti na gumb "Solve problem" što će pokušati riješiti linearni program i ispisati na zaslon poruku o uspješnosti. U slučaju da je za linearni program nađeno rješenje tada klikom na gumb "Display result" možemo prikazati najveću vrijednost za funkciju cilja te rješenje za koje se ta vrijednost postiže što možemo vidjeti zaokruženo u crvenom na slici 14.




Slika 14: Poruka uspješnosti i rješenje za linearni program

Na vrhu stranice, klikom na gumb "Manufacturing problem" korisnik se može prebaciti na obrazac za problem proizvodnje.




Slika 15: Obrazac za problem proizvodnje

Pri vrhu se nalazi opis samog problema proizvodnje te što pojedini stupac ili redak predstavlja.

Ukupan broj proizvoda i resursa može mijenjati preko ovih polja pored labela "Number of products" i "number of resources". Broj proizvoda i resursa ne može biti manji od 2, te broj proizvoda uvijek mora biti veći ili jednak broju resursa.

Ispod toga se nalazi tablica u koju korisnik upisuje podatke. U prvi redak se upisuje imena proizvoda, u zadnji redak se upisuju cijene proizvoda, u prvi stupac se upisuju imena resursa te se u zadnji stupac upisuje ukupno količinu tog resursa koju imam na raspolaganju. U stupac za dani proizvod pišemo koliko resursa u tom retku je potrebno kako bi ga mogli proizvesti.

Nakon upisivanje korisnik prati isti postupak kao kod obrasca maksimiziranja linearnog problema, prvo klikne na gum "Export linear program" te kad se pojavi vijest o uspješnom eksportiranju klinke gumb "Solve problem" te ako se pojavi vijest o uspješnom prolaženju rješenja može kliknuti na gumb "Display result" kako bi vidjeli najveću zaradu za dane podatke i koliko kojeg proizvoda moraju proizvesti kako bi došli do te zarade.

# 5. Zaključak

Linearno programiranje je nastalo prije 90 godina iz potrebe za poboljšanjem postojećih procesa. Riječ je o području čiji su osnivači počeli razvijati teoriju nakon što su stekli određenu razinu iskustva u industriji tijekom vremena kada je razvoj u svim područjima znanosti bi znatno ubrzan. Teorija sama je zrela te je dovoljna određena razina predznanja iz linearne algebre kako bi se objasnila.

Simpleks metoda se nastavljala razvijati nakon Dantzigove objave te je jedan od glavnih utjecaj daljeg razvoja bila pojava računala. Iako je pojava računala dozvolila ubrzan razvoj u svim područjima matematike, linearno programiranje je ispalo pogotovo pogodno za implementaciju u kodu te je smjer razvoja simpleks metode težio lakšoj i bržoj programskoj implementaciji. Brzina rješavanja linearnih programa i veličina rješivih linearnih programa su rasla.

Dio programa koji je napisan u sklopu ovog završnog rada povezan sa implementacijom simpleks metode, čiji su koraci skoro doslovno implementirani, je efikasan te je moguće na modernom hardveru u prihvatljivom vremenu riješiti linearne programe srednje složenosti. Drugi dio programa je druga priča.

Formulirati problem iz pravog svijeta kao linearni program nije trivijalno. Potrebna je odrađena razina iskustva u području koje ima probleme koji se mogu izraziti linearnim programima i znanje iz samog linearnog programiranja, te bi za osmišljavanje obrazaca bila ideal osoba koji ima ekspertizu u oba područja. Za ovaj završni rad bi bilo bolje da se znatno više vremena provelo proučavanjem problemskih domena kako bi se alati koji su razvijeni mogli bolji iskoristiti, ali bi takav zahvat vrlo vjerojatno bio izvan raspona jednog završnog rada.

Nasreću, arhitektura programske implementacija dozvoljava razvoj novih, smislenijih obrazaca u budućnosti.

# Popis literature


[1] D. Bertsimas i J. N. Tsitsiklis, *Introduction to Linear Optimization*, en. Athena Scientific, 1997., Google-Books-ID: GAFsQgAACAAJ, ISBN: 978-1-886529-19-9.

[2] D. G. Luenberger i Y. Ye, *Linear and Nonlinear Programming* (International Series in Operations Research & Management Science), English, Fourth Edition. Springer, sv. 228.

[3] G. B. Dantzig, „Reminiscences about the origins of linear programming,” en, *Operations Research Letters*, sv. 1, br. 2, str. 43–48, travanj 1982., ISSN: 0167-6377. DOI: 10.1016/0167-6377(82)90043-8. adresa: https://www.sciencedirect.com/science/article/pii/0167637782900438 (pogledano 17. 6. 2023.).

# Popis slika

# Popis tablica