UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Finančna matematika – 1. stopnja

Tina Janša

# Spletna aplikacija za linearno programiranje

Delo diplomskega seminarja

Mentor: izred. prof. dr. Andrej Bauer
Somentor: asist. dr. Matija Pretnar

Ljubljana, 2012

Kazalo

**Spletna aplikacija za linearno programiranje**

Povzetek

Delo diplomskega seminarja predstavlja spletno aplikacijo, ki uporabniku brez matematičnega predznanja omogoča reševanje linearnih programov. V delu je najprej prikazan primer uporabe aplikacije. Nato sledi kratek pregled reševanja linearnih programov s simpleksno metodo in celoštevilskih linearnih programov z metodo razveji in omeji. Pred opisom strukture in delovanja aplikacije so predstavljena osnovna orodja za izdelavo spletnih aplikacij, kot so Bottle, Jinja2 in PuLP. Opisane so tudi možne izboljšave, ki bi naredile aplikacijo lažjo za uporabo.

**Web application for linear programming**

Abstract

The main topic of the thesis is a web application for solving linear programs, designed for people without proper mathematical knowledge. In the beginning an example of application use is described. Then there is an overview of how to solve linear programs with simplex algorithm and integer linear programs with branch and bound method. Later on the tools for making web applications such as Bottle, Jinja2 and PuLP are presented. The work also describes the structure of the application and how it works. At the end possible solutions for easier use are presented.

**Math. Subj. Class. (2010):** 68 Computer science, 90 Operations research, mathematical programming

**Ključne besede:** linearno programiranje, celoštevilsko linearno programiranje, spletna aplikacija, uporabniški vmesnik, Python, Bottle, Jinja2, PuLP

**Keywords:** linear programming, integer linear programming, web application, user interface, Python, Bottle, Jinja2, PuLP

# 1. Uvod

Namen dela diplomskega seminarja je izdelati spletno aplikacijo, to je program, do katerega dostopamo preko spletnega brskalnika, s pomočjo katere rešujemo linearne optimizacijske probleme. Namenjena bo predvsem uporabnikom, ki nimajo potrebnega matematičnega znanja za reševanje in zapis linearnih programov.

## 1.1. Linearno programiranje.

*Linearni program* je optimizacijska naloga, s katero iščemo optimum linearne funkcije (*kriterijska funkcija*) pri danih linearnih neenačbah (*omejitve*).

Tipična problema, ki se ju lahko reši z uporabo linearnega programiranja, sta proizvodni in prehrambni problem. V proizvodnem problemu nastopa podjetje, ki ima na voljo določene količine produkcijskih faktorjev (npr. material, ure dela, denar ...), s katerimi proizvaja razne izdelke. Za vsak izdelek ve, koliko posameznega produkcijskega faktorja porabi pri proizvodnji in kakšna je tržna cena posameznega izdelka. Podjetje zanima, kako naj planira proizvodnjo, da bo doseglo največji dobiček. Pri prehrambnem problemu nas zanima, koliko posameznih živil je potrebno nakupiti, da zadostimo našim biološkim potrebam po določeni količini sestavin, pri čemer naj bodo stroški nakupa najmanjši. Vsako živilo ima določeno biokemično strukturo in določeno ceno.

## 1.2. Primer.

Poglejmo si primer, ki se ga da rešiti z linearnim programiranjem. Želimo pripraviti zajtrk iz ovsenih in koruznih kosmičev, rozin, banane in jogurta. V spodnji tabeli je podano, koliko kilokalorij, gramov maščob, beljakovin, ogljikovih hidratov in vlaknin vsebuje 100 g posameznega živila [2].

| | ovseni kosmiči | koruzni kosmiči | rozine | banana | jogurt |
|---|---|---|---|---|---|
| kalorije | 372 | 368 | 246 | 79 | 61 |
| maščobe | 7.5 | 1.6 | 0 | 0.3 | 3.2 |
| beljakovine | 8 | 8.6 | 1.1 | 1.1 | 3.5 |
| ogljikovi hidrati | 72.8 | 85.1 | 64.4 | 19.9 | 4.7 |
| vlaknine | 7 | 11 | 6.8 | 3.4 | 0 |

Zanima nas, koliko posameznega živila naj uporabimo, če zajtrk ne sme vsebovati več kot 18 g maščob ali več kot 70 g ogljikovih hidratov, mora pa vsebovati vsaj 12 g beljakovin in 7 g vlaknin. Želimo tako kombinacijo živil, da bomo zaužili minimalno količino kalorij. Na voljo imamo 200 g jogurta, zajtrk pa naj bo sestavljen iz vsaj 10 g rozin, 10 g koruznih kosmičev in iz 10 g ovsenih kosmičev.

Problem zapišemo v obliki linearnega programa. Spremenljivke so količine živil v 100 g: ovseni kosmiči ($x_1$), koruzni kosmiči ($x_2$), rozine ($x_3$), banana ($x_4$), jogurt ($x_5$). Omejitve so pogoji, koliko katerih sestavin (maščob, beljakovin, ogljikovih hidratov, vlaknin) naj bi zaužili z zajtrkom in koliko posameznih živil naj zajtrk vsebuje. Kriterijska funkcija je število kalorij, ki jih bomo zaužili s pripravljenim zajtrkom. Iščemo minimum te funkcije, ker želimo zajtrk s čim manj kalorijami. Za spremenljivke zahtevamo nenegativnost, saj ne moremo sestaviti zajtrka iz negativne količine posameznih živil.

$$
\begin{aligned}
7.5x_1 + 1.6x_2 + 0.3x_4 + 3.2x_5 &\leq 18 && \text{(maščobe)}\\
8x_1 + 8.6x_2 + 1.1x_3 + 1.1x_4 + 3.5x_5 &\geq 12 && \text{(beljakovine)}\\
72.8x_1 + 85.1x_2 + 64.4x_3 + 19.9x_4 + 4.7x_5 &\leq 70 && \text{(ogljikovi hidrati)}\\
7x_1 + 11x_2 + 6.8x_3 + 3.4x_4 &\geq 7 && \text{(vlaknine)}\\
x_1 &\geq 0.1 && \text{(ovseni kosmiči)}\\
x_2 &\geq 0.1 && \text{(koruzni kosmiči)}\\
x_3 &\geq 0.1 && \text{(rozine)}\\
x_5 &\leq 0.2 && \text{(jogurt)}\\
x_1,\ x_2,\ x_3,\ x_4,\ x_5 &\geq 0 &&\\
\min 372x_1 + 368x_2 + 246x_3 + 79x_4 + 61x_5 && && \text{(število kalorij)}
\end{aligned}
$$

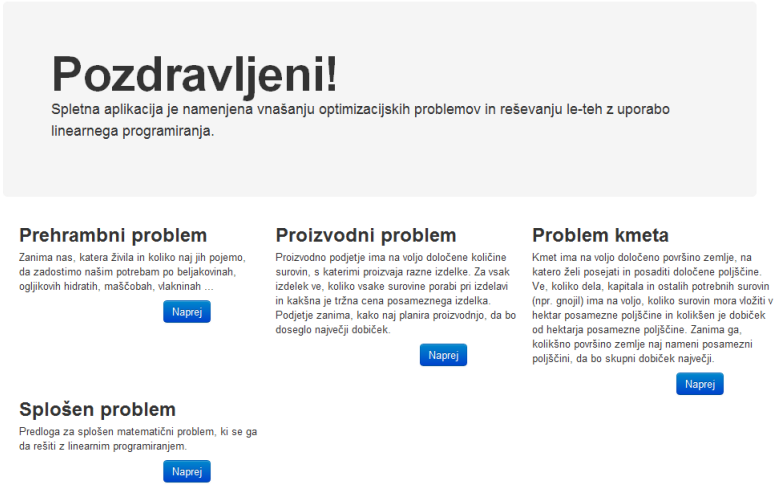
Ko dani linearni program rešimo, dobimo rešitev:

$$
\begin{aligned}
x_1 &= 0.1\\
x_2 &= 0.45\\
x_3 &= 0.1\\
x_4 &= 0.195\\
x_5 &= 2.0\\
z &= 365.04
\end{aligned}
$$

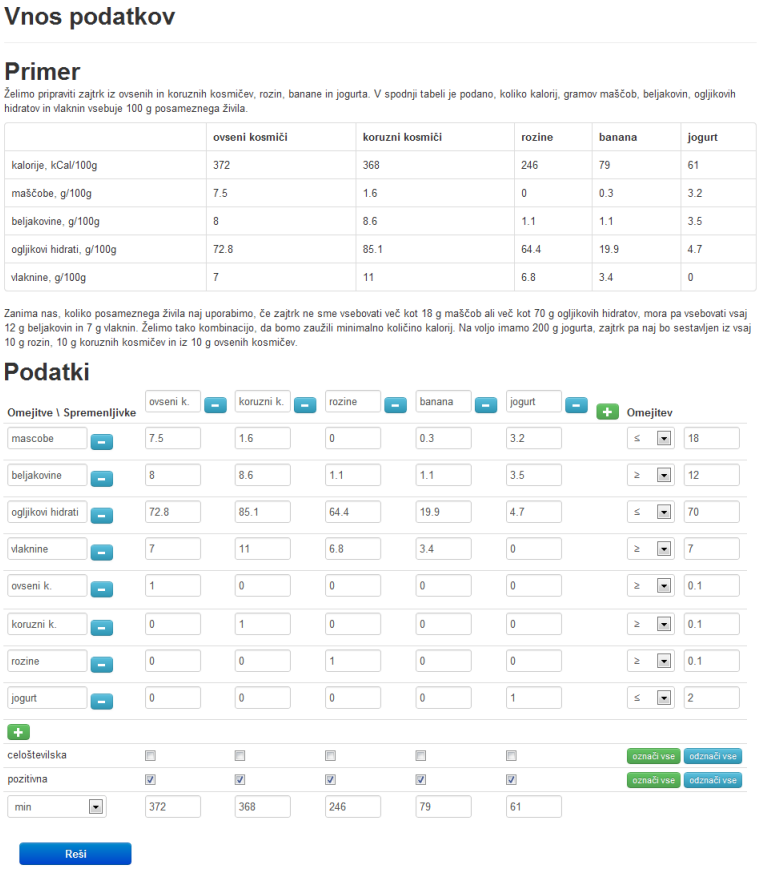
Rešitev problema za pripravo zajtrka je naslednja: najmanj kalorij, 365.04, bomo zaužili, če si bomo pripravili zajtrk iz 10 g ovsenih kosmičev, 45 g koruznih kosmičev, 10 g rozin, 19.5 g banane in 200 g jogurta.

Linearni program lahko rešimo s pomočjo metode simpleksov, ki je opisana v razdelku 2.1. Ker večina ljudi, ki niso matematiki, ne pozna te metode, obstajajo računalniški programi, s pomočjo katerih lahko zapišemo in rešimo linearni program. Eden izmed takih programov je knjižnica PuLP za Python, ki je opisana v razdelku 3.4. Vendar je tudi ob uporabi knjižnice potrebno zapisati problem v ustrezni matematični obliki in poznati Python. Zato sem naredila spletni vmesnik, ki bo ljudem brez potrebnega znanja omogočil, da zapišejo in rešijo svoj optimizacijski problem s pomočjo linearnega programiranja. S pomočjo vzorčnih primerov sem poskusila zagotoviti, da bo uporaba aplikacije čim bolj preprosta. Po vnosu podatkov aplikacija s pomočjo knjižnice PuLP reši problem in uporabniku prikaže rešitev.

1.3. **Uporabniški vmesnik.** Uporabnik dostopa do aplikacije preko spletnega brskalnika. Na začetni strani je na voljo več različnih vzorčnih problemov, ki se jih da rešiti s pomočjo linearnega programiranja. Uporabnik izbere tistega, ki je najbliže njegovemu problemu (Slika 1). Če želimo rešiti primer, ki je opisan v razdelku 1.2, bomo izbrali vzorec 'Prehrambni problem'. Vzorec je sestavljen iz opisa vzorčnega problema in tabele, ki je namenjena vnosu podatkov problema. Na začetku so v tabelo vneseni podatki vzorčnega problema. Naš primer je eden izmed vzorčnih problemov, zato so podatki že vneseni v tabelo (Slika 2). Uporabnik lahko po potrebi razširi ali skrči tabelo (doda ali odstrani vrstice/omejitve in stolpce/spremenljivke), nato vnese podatke svojega problema (Slika 3). Za vsako izmed spremenljivk lahko

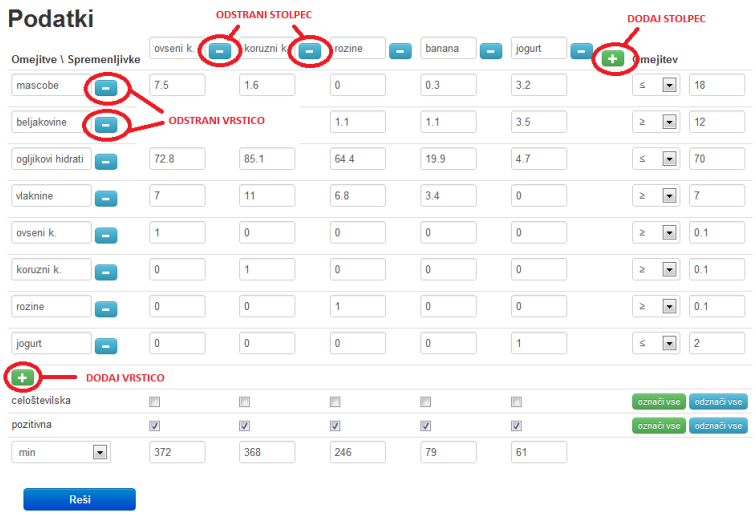



Slika 1. Uporabnik na prvi strani izbere ustrezen vzorec.

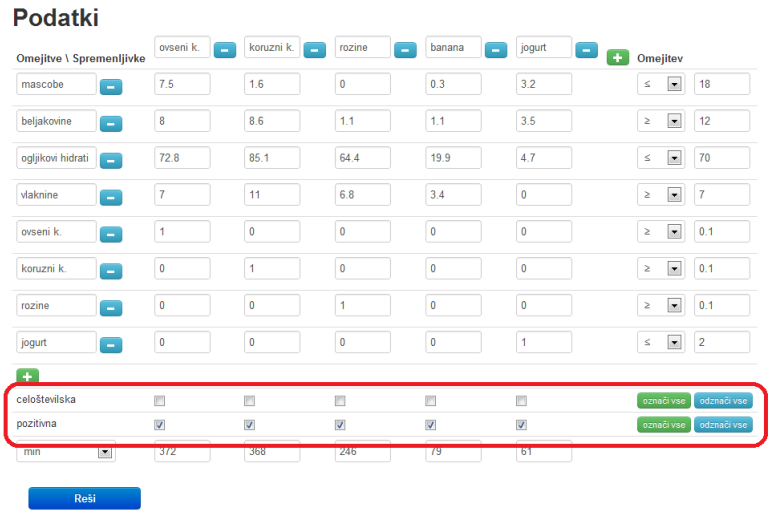



Slika 2. Vzorec vsebuje opis vzorčnega problema in tabelo za vnašanje podatkov.

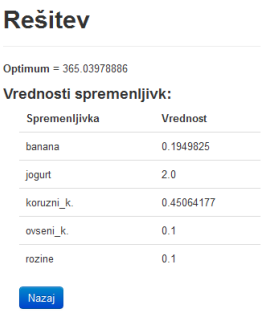
izbere, ali mora biti celoštevilska in/ali pozitivna (Slika 4). Ko uporabnik vnese svoj problem, klikne na gumb 'Reši' in dobi rešitev problema, če optimalna rešitev obstaja (Slika 5). Če optimalna rešitev danega problema ne obstaja, se mu izpiše




Slika 3. Uporabnik lahko spremeni velikost tabele z izbiro ustreznega gumba.




Slika 4. Uporabnik lahko izbere, ali morajo biti spremenljivke celoštevilske in pozitivne.




Slika 5. Rešitev problema priprave zajtrka.

sporočilo z razlago, zakaj je do tega prišlo. V rešitvi so zapisani optimum (minimum oz. maksimum) kriterijske funkcije ter optimalne vrednosti vseh nastopajočih spremenljivk.

## 2. Linearno programiranje

Pred opisom spletne aplikacije se spomnimo nekaj osnov o linearnih programih. Za podrobnejši opis glej [3] in [4].

V splošnem je linearni program naslednje oblike:

$$\begin{aligned} \text{iščemo optimum} \quad & \underline{c}^T\underline{x} \quad (\min \text{ ali } \max) \\ \text{pri pogojih} \quad & A\underline{x} \leq \underline{b} \\ & A'\underline{x} \geq \underline{b}' \\ & A''\underline{x} = \underline{b}'' \\ & x_i \geq 0 \text{ za nekatere } i \end{aligned}$$

kjer so $A$, $A'$, $A''$ matrike koeficientov, $\underline{b}$, $\underline{b}'$, $\underline{b}''$ vektorji prostih členov, $\underline{x}$ vektor spremenljivk ter $\underline{c}$ vektor koeficientov kriterijske funkcije.

Vsak splošni linearni program lahko pretvorimo v *standardno obliko*:

$$\begin{aligned} \text{iščemo max} \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{pri pogojih} \quad & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \\ & \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m \\ & x_1,\, x_2, \ldots, x_n \geq 0 \end{aligned}$$

V matrični obliki standardni linearni program zapišemo kot:

$$\begin{aligned} \max \quad & \underline{c}^T\underline{x} \\ \text{p.p.} \quad & A\underline{x} \leq \underline{b} \\ & \underline{x} \geq \underline{0}, \\ \text{kjer je } & A \in \mathbb{R}^{m\times n},\ \underline{b} \in \mathbb{R}^m,\ \underline{c} \in \mathbb{R}^n \end{aligned}$$

Linearni program pretvorimo v standardno obliko na naslednji način:

- namesto $\min \underline{c}^T\underline{x}$ iščemo $-\max(-\underline{c}^T\underline{x})$;
- neenačbe oblike $A'\underline{x} \geq \underline{b}'$ pomnožimo z $-1$, da dobimo $-A'\underline{x} \leq -\underline{b}'$;
- enačbe $A''\underline{x} = \underline{b}''$ zapišemo z dvema neenačbama: $A''\underline{x} \leq \underline{b}''$ in $-A''\underline{x} \leq -\underline{b}''$;
- spremenljivke, ki nimajo pogoja nenegativnosti, razcepimo na razliko dveh novih nenegativnih spremenljivk: $x_i = x_i^+ - x_i^-,\ x_i^+,\ x_i^- \geq 0$.

*Dopustna rešitev* linearnega programa v standardni obliki je vektor $\underline{x}$, za katerega velja $A\underline{x} \leq \underline{b}$. Dopustna rešitev, v kateri kriterijska funkcija zavzame maksimum, se imenuje *optimalna rešitev*.

Linearni program lahko nima rešitve, če je nedopusten (t.j. $\{\underline{x} \in \mathbb{R}^n; A\underline{x} \leq \underline{b}\} = \emptyset$) ali neomejen (t.j. za vsak $\alpha \in \mathbb{R}$ obstaja $\underline{x} \in \{\underline{x} \in \mathbb{R}^n; A\underline{x} \leq \underline{b}\}$, da velja $\underline{c}^T\underline{x} > \alpha$). Če linearni program ni niti nedopusten niti neomejen, ima optimalno rešitev.

2.1. **Metoda simpleksov.** Linearni program v standardni obliki se običajno rešuje po metodi simpleksov:

(1) Z uvedbo dopolnilnih spremenljivk (za vsako od $m$ neenačb eno), vse neenačbe dopolnimo do enačb. Vrednost dopolnilne spremenljivke v $i$-ti enačbi je enaka razliki med desno in levo stranjo $i$-te neenačbe. Za prvotne in dopolnilne spremenljivke zahtevamo nenegativnost. Dobimo razširjen linearni program, ki ga zapišemo v obliki slovarja:

$$\begin{aligned} x_{n+1} &= b_1 - a_{11}x_1 - a_{12}x_2 - \cdots - a_{12}x_2 \\ &\vdots \\ x_{n+m} &= b_m - a_{m1}x_1 - a_{m2}x_2 - \cdots - a_{mn}x_n \\ z &= c_1x_1 + c_2x_2 + \cdots + c_nx_n \end{aligned}$$

Spremenljivke na levi strani slovarja se imenujejo bazne spremenljivke, tiste na desni strani slovarja pa nebazne spremenljivke, $z$ je vrednost kriterijske funkcije (funkcionala). Vrednost nebaznih spremenjivk je 0, vrednost baznih spremenljivk in funkcionala pa določa prosti člen v ustrezni enačbi. Rešitev razširjenega linearnega programa se imenuje bazna rešitev. Če bazne spremenljivke zadoščajo pogoju nenegativnosti, se rešitev imenuje bazno dopustna rešitev.

(2) Postopek reševanja:
- Izberemo eno izmed nebaznih spremenljivk s pozitivnim koeficientom v funkcionalu (običajno tisto, ki ima največji koeficient). Imenujemo jo vstopajoča spremenljivka.
- Pogoji nenegativnosti za bazne spremenljivke omejujejo vstopajočo spremenljivko. Izberemo tisto bazno spremenljivko, katere pogoj je najbolj omejujoč. Izbrano spremenljivko imenujemo izstopajoča spremenljivka.
- Vstopajoča spremenljivka postane bazna spremenljivka, izstopajoča pa nebazna spremenljivka. Enačbe preuredimo tako, da so bazne spremenljivke na levi strani enačb, nebazne pa na desni.

(3) Postopek pod točko 2 ponavljamo, dokler ni več nobene spremenljivke, ki bi imela pozitiven koeficient v funkcionalu. Tedaj je trenutna bazno dopustna rešitev optimalna. Vrednosti prvotnih spremenljivk sestavljajo optimalno rešitev prvotnega linearnega programa, maksimum kriterijske funkcije pa je enak vrednosti funkcionala $z$.

Kadar je v slovarju kakšen prosti člen negativen, je slovar nedopusten. V takem primeru uporabimo dvofazno simpleksno metodo, kjer s pomočjo prve faze dobimo dopusten slovar. Za opis dvofazne simpleksne metode glej npr. [4].

2.2. **Primer uporabe metode simpleksov.** Imamo linearni program, ki ga želimo rešiti po metodi simpleksov:

$$\begin{aligned} \max \quad & 3x_1 + 5x_2 + 4x_3 \\ \text{p.p.} \quad & x_1 + x_2 + x_3 \leq 50 \\ & 3x_1 + 4x_2 + 5x_3 \leq 250 \\ & 10x_1 + 15x_2 + 12x_3 \leq 600 \end{aligned}$$

Za vsako od neenačb uvedemo dopolnilno spremenljivko in vse neenačbe dopolnimo do enačb. Dobimo naslednji slovar:

$$\begin{aligned} x_4 &= 50 - x_1 - x_2 - x_3 \\ x_5 &= 250 - 3x_1 - 4x_2 - 5x_3 \\ x_6 &= 600 - 10x_1 - 15x_2 - 12x_3 \\ z &= 3x_1 + 5x_2 + 4x_3 \end{aligned}$$

V našem primeru so bazne spremenljivke $x_4$, $x_5$ in $x_6$, nebazne pa $x_1$, $x_2$ in $x_3$. Izmed nebaznih spremenljivk izberemo vstopajočo spremenljivko, na primer $x_2$, ker ima največji pozitiven koeficient v funkcionalu $z$.

Pogoji nenegativnosti za bazne spremenljivke omejujejo povečanje vrednosti vstopajoče spremenljivke $x_2$:

$$\begin{aligned} x_4 &= 50 - x_2 \geq 0 \Rightarrow x_2 \leq 50 \\ x_5 &= 250 - 4x_2 \geq 0 \Rightarrow x_2 \leq 62.5 \\ x_6 &= 600 - 15x_2 \geq 0 \Rightarrow x_2 \leq 40 \end{aligned}$$

Najbolj omejujoč je zadnji pogoj, zato za izstopajočo spremenljivko izberemo $x_6$. Slovar preuredimo tako, da so nove bazne spremenljivke na levi strani enačb, nebazne pa na desni (iz 3. enačbe slovarja izrazimo $x_2$ in jo vstavimo v ostale enačbe). Dobimo nov slovar:

$$\begin{aligned} x_2 &= 40 - \frac{2}{3}x_1 - \frac{4}{5}x_3 - \frac{1}{15}x_6 \\ x_4 &= 10 - \frac{1}{3}x_1 - \frac{1}{5}x_3 + \frac{1}{15}x_6 \\ x_5 &= 90 - \frac{1}{3}x_1 - \frac{9}{5}x_3 + \frac{4}{15}x_6 \\ z &= 200 - \frac{1}{3}x_1 - 0x_3 - \frac{1}{3}x_6 \end{aligned}$$

S tem korakom končamo, ker nobena spremenljivka nima pozitivnega koeficienta v funkcionalu. Trenutna bazno dopustna rešitev je optimalna. Nebazne spremenljivke imajo vrednost 0: $x_1 = x_3 = x_6 = 0$. Bazne spremenljivke in vrednost funkcionala določa prosti člen v ustrezni enačbi: $x_2 = 40$, $x_4 = 10$, $x_5 = 90$, $z = 200$. Rešitev začetnega problema je: maksimum funkcije, 200, je dosežen pri $x_1 = 0$, $x_2 = 40$, $x_3 = 0$.

2.3. **Celoštevilsko linearno programiranje.** Pogosto v linearnih programih nastopa dodatna zahteva, da spremenljivke (vse ali le nekatere) lahko zavzamejo le celoštevilske vrednosti. Tak linearni program se imenuje celoštevilski linearni program (*integer linear program*). Če zahtevamo celoštevilskost le za nekatere spremenljivke, ga imenujemo mešani celoštevilski linearni program (*mixed integer linear program*). Včasih spremenljivke lahko zavzamejo le vrednosti 0 ali 1. Takemu linearnemu programu rečemo binarni celoštevilski linearni program (*binary integer linear program*).

Mešani celoštevilski linearni program, kjer za prvih $I$ spremenljivk zahtevamo, da zavzamejo samo celoštevilske vrednosti, ostale pa lahko zavzamejo poljubne realne

vrednosti, zapišemo v standardni obliki:

$$\begin{aligned} \text{iščemo maksimum} \quad & Z = \sum_{j=1}^{n} c_j x_j \\ \text{pri pogojih} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i \text{ za } i = 1, \ldots, \ m \\ & x_j \geq 0 \text{ za } j = 1, \ldots, \ n \\ & x_j \text{ je celo število za } j = 1, \ldots, I, \ I \leq n \end{aligned}$$

2.4. **Razveji in omeji.** Ena izmed metod za reševanje celoštevilskih linearnih programov je *razveji in omeji* (*branch and bound*). Osnovna ideja je, da začetni problem razdelimo na manjše podprobleme, ki so laže rešljivi. Podprobleme dobimo tako, da dopustno množico za začetni problem razdelimo na manjše podmnožice. Nato poiščemo oceno najboljše rešitve na dani podmnožici in jo izločimo, če ugotovimo, da ne more vsebovati optimalne rešitve za začetni problem.

Ocena najboljše rešitve za dani podproblem je vrednost funkcionala navadnega linearnega programa, ki ga dobimo iz celoštevilskega linearnega programa, tako da izpustimo pogoje celoštevilskosti. Običajno navadni linearni program rešimo po metodi simpleksov.

Algoritem je sestavljen iz več korakov. Najprej rešimo navaden linearni program, ki ga dobimo iz začetnega problema. Vrednost funkcionala navadnega linearnega programa je zgornja meja za naš problem. Če je optimalna rešitev celoštevilska, potem je to tudi rešitev začetnega problema. Če pa optimalna rešitev vsebuje spremenljivko, npr. $x_i$, katere vrednost $x_i^*$ ni celo število, a bi morala biti, ponavljamo naslednje korake:

(1) **Razveji:** Izberemo enega izmed neizločenih podproblemov (običajno tistega, ki ima največjo vrednost funkcionala pripadajočega navadnega linearnega programa) in eno od spremenljivk, ki bi morala biti celoštevilska, a v optimalni rešitvi navadnega linearnega programa nima celoštevilske vrednosti (običajno tisto, ki je najdlje od celega števila). Naj bo ta spremenljivka $x_i$ in naj bo $x_i^*$ njena vrednost v optimalni rešitvi. Podproblem razvejimo na dva nova podproblema, enega z dodatno zgornjo omejitvijo $x_i \leq \lfloor x_i^* \rfloor$ in drugega z dodatno spodnjo omejitvijo $x_i \geq \lfloor x_i^* \rfloor + 1$. Zaradi dodatnih omejitev $x_i^*$ ni rešitev nobenega izmed dobljenih podproblemov.

(2) **Omeji:** Za vsakega izmed novih podproblemov izračunamo oceno najboljše rešitve (rešimo pripadajoč navaden linearni program). Največja izmed ocen dosedanjih podproblemov, ki imajo celoštevilske rešitve, je spodnja meja za začetni problem.

Podproblem lahko izločimo iz nadaljnjega vejanja, če:

- je vrednost funkcionala pripadajočega navadnega linearnega programa manjša od spodnje meje problema;
- pripadajoči navadni linearni program nima dopustne rešitve;
- ima pripadajoči navadni linearni program celoštevilsko rešitev. Če je vrednost funkcionala večja od spodnje meje, postane optimalna rešitev navadnega linearnega programa trenutna najboljša rešitev za začetni problem, vrednost funkcionala pa nova spodnja meja.

Koraka ponavljamo, dokler ni več nobenega podproblema, ki bi ga lahko razvejali. Tedaj je trenutna najboljša rešitev optimalna rešitev začetnega celoštevilskega linearnega programa. Če trenutna najboljša rešitev ne obstaja, začetni celoštevilski linearni program nima dopustne rešitve.

2.5. **Primer uporabe razveji in omeji.** Imamo naslednji celoštevilski linearni program, ki ga želimo rešiti po metodi razveji in omeji [5]:

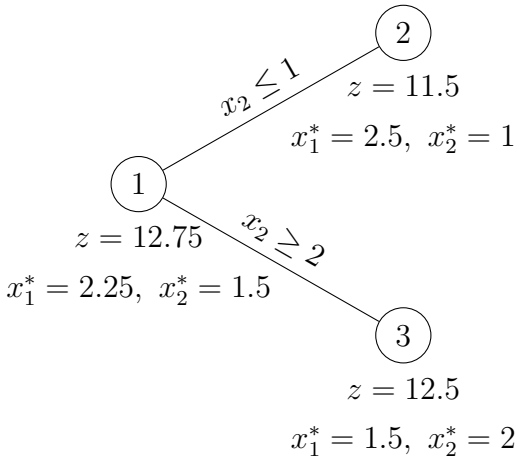
$$\begin{aligned} (1)\ \max\quad & 3x_1 + 4x_2 \\ \text{p.p.}\quad & 2x_1 + x_2 \leq 6 \\ & 2x_1 + 3x_2 \leq 9 \\ & x_1,\ x_2 \geq 0 \\ & x_1,\ x_2 \text{ celoštevilski} \end{aligned}$$

Najprej rešimo navaden linearni program brez pogoja celoštevilskosti (npr. po metodi simpleksov) in dobimo rešitev: $x_1^* = 2.25,\ x_2^* = 1.5,\ z = 12.75$. Ker vrednosti obeh spremenljivk nista celoštevilski, a bi morali biti, izberemo eno, na katero bomo problem razvejili. Ker je $x_2^*$ bolj oddaljena od celega števila, razvejimo najprej na $x_2$. Dobimo dva podproblema, prvega z dodatno spodnjo omejitvijo $x_2 \geq 2$ in drugega z dodatno zgornjo omejitvijo $x_2 \leq 1$:

$$\begin{aligned} (2)\ \max\quad & 3x_1 + 4x_2 \\ \text{p.p.}\quad & 2x_1 + x_2 \leq 6 \\ & 2x_1 + 3x_2 \leq 9 \\ & x_2 \leq 1 \\ & x_1,\ x_2 \geq 0 \\ & x_1,\ x_2 \text{ celoštevilski} \end{aligned}$$

$$\begin{aligned} (3)\ \max\quad & 3x_1 + 4x_2 \\ \text{p.p.}\quad & 2x_1 + x_2 \leq 6 \\ & 2x_1 + 3x_2 \leq 9 \\ & x_2 \geq 2 \\ & x_1,\ x_2 \geq 0 \\ & x_1,\ x_2 \text{ celoštevilski} \end{aligned}$$

Če rešimo ta dva linearna programa brez upoštevanja pogoja celoštevilskosti, dobimo, da je rešitev podproblema (2): $x_1^* = 2.5,\ x_2^* = 1,\ z = 11.5$, rešitev podproblema (3) pa: $x_1^* = 1.5,\ x_2^* = 2,\ z = 12.5$.

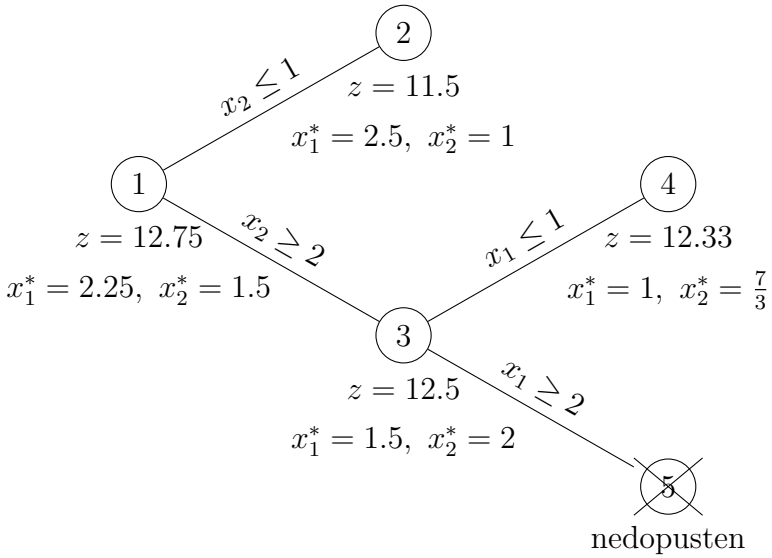



Ker imata oba podproblema necelo\v{s}tevilski re\v{s}itvi, moramo razvejati oba. Najprej izberemo podproblem (3), ker ima ve\v{c}jo vrednost funkcionala $z$. Razvejimo na $x_1$, saj je vrednost $x_2$ že celo število. Dodani omejitvi za nova podproblema sta $x_1 \geq 2$ in $x_1 \leq 1$:

$$
\begin{aligned}
(4)\ \max \quad & x_1 + 4x_2 \\
\text{p.p.} \quad & 2x_1 + x_2 \leq 6 \\
& 2x_1 + 3x_2 \leq 9 \\
& x_2 \geq 2 \\
& x_1 \leq 1 \\
& x_1,\ x_2 \geq 0 \\
& x_1,\ x_2 \text{ celoštevilski}
\end{aligned}
$$

$$
\begin{aligned}
(5)\ \max \quad & 3x_1 + 4x_2 \\
\text{p.p.} \quad & 2x_1 + x_2 \leq 6 \\
& 2x_1 + 3x_2 \leq 9 \\
& x_2 \geq 2 \\
& x_1 \geq 2 \\
& x_1,\ x_2 \geq 0 \\
& x_1,\ x_2 \text{ celoštevilski}
\end{aligned}
$$

Re\v{s}itev podproblema (4) brez upoštevanja pogoja celoštevilskosti je: $x_1^* = 1$, $x_2^* = \frac{7}{3}$ in $z = 12.33$. Podproblem (5) je nedopusten, zato ga lahko izlo\v{c}imo iz nadaljnjega vejanja.



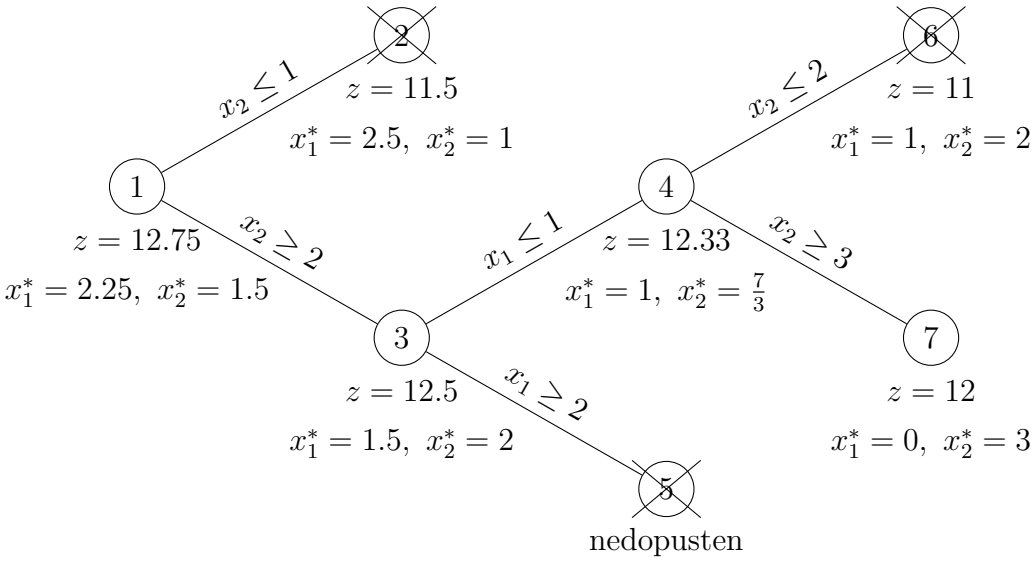

V naslednjem koraku lahko razvejimo podproblem (2) ali podproblem (4). Izberemo (4), ker ima večjo vrednost funkcionala $z$. Razvejimo na $x_2$, saj je $x_1$ že celo število. Dodani omejitvi za nova podproblema sta $x_2 \leq 2$ in $x_2 \geq 3$:

$$
\begin{aligned}
(6)\ \max \quad & 3x_1 + 4x_2 \\
\text{p.p.} \quad & 2x_1 + x_2 \leq 6 \\
& 2x_1 + 3x_2 \leq 9 \\
& x_2 \geq 2 \\
& x_1 \leq 1 \\
& x_2 \leq 2 \\
& x_1,\ x_2 \geq 0 \\
& x_1,\ x_2 \text{ celoštevilski}
\end{aligned}
$$

$$
\begin{aligned}
(7)\ \max \quad & 3x_1 + 4x_2 \\
\text{p.p.} \quad & 2x_1 + x_2 \leq 6 \\
& 2x_1 + 3x_2 \leq 9 \\
& x_2 \geq 2 \\
& x_1 \leq 1 \\
& x_2 \geq 3 \\
& x_1,\ x_2 \geq 0 \\
& x_1,\ x_2 \text{ celoštevilski}
\end{aligned}
$$

Rešitev podproblema (6) brez upoštevanja pogoja celoštevilskosti je: $x_1^* = 1$, $x_2^* = 2$, $z = 11$. Ker ima pripadajoči navadni linearni program celoštevilsko rešitev, postane vrednost funkcionala ($z = 11$) spodnja meja za začetni problem. Vsak podproblem z vrednostjo funkcionala manj kot 11 lahko izločimo. Ker noben podproblem nima vrednosti funkcionala manj kot 11, ne izločimo ničesar.

Rešitev podproblema (7) brez upoštevanja pogoja celoštevilskosti je: $x_1^* = 0$, $x_2^* = 3$, $z = 12$. Tudi ta podproblem ima celoštevilsko rešitev. Ker je vrednost funkcionala večja kot 11, je nova spodnja meja 12 (vrednost funkcionala pripadajočega navadnega linearnega programa). Vsak podproblem z vrednostjo funkcionala $z$ manj kot 12 lahko izločimo iz nadaljnjega vejanja. Torej izločimo podproblema (6) in (2).




Ker ni več nobenega podproblema, ki bi ga lahko razvejali, je rešitev podproblema (7) tudi rešitev začetnega problema: $x_1^* = 0$, $x_2^* = 3$ in $z = 12$.

## 3. Sestavni deli spletne aplikacije

Pred opisom spletne aplikacije si oglejmo orodja, ki nam omogočajo izdelavo spletnega vmesnika. Pri izdelavi aplikacije sem poleg Pythona in HTML-ja uporabila različne že obstoječe knjižnice, kot so Bottle, Jinja2 in Pulp, za lepši izgled spletnih strani pa CSS stil Bootstrap.

3.1. **HTTP.** Komunikacija s strežnikom poteka preko HTTP-ja [6]. To je spletni protokol (HyperText Transfer Protocol), v katerem stranka pošlje zahtevo strežniku, ki ji vrne odgovor. V našem primeru je stranka spletni brskalnik, strežnik pa spletna aplikacija.

Zahteva stranke vsebuje HTTP metodo in naslov zahtevane strani ali datoteke. Najbolj pogosta HTTP metoda je GET, ki se uporablja za pridobitev vsebine strani ali datoteke. Metoda POST se uporablja za pošiljanje podatkov strežniku, največkrat za pošiljanje podatkov iz HTML obrazcev.

Odgovor strežnika vsebuje status 200 OK in vsebino zahtevane spletne strani ali datoteke. Če zahtevana spletna stran ne obstaja, strežnik vrne status 404 Not Found. Pogost status je tudi 500 Server Error, ki pove, da je prišlo do napake na strežniku.

Zahteve so med sabo neodvisne, zato mora stranka ob vsaki zahtevi strežniku poslati vse potrebne podatke. Če na primer v spletnem iskalniku izberemo povezavo do naslednje strani zadetkov, strežnik poleg zahteve ponovno dobi podatke o iskanem geslu.

3.2. **Bottle.** Bottle [7] je knjižnica za izdelavo spletnih aplikacij (vmesnikov) v Pythonu. Knjižnica omogoča zagon HTTP strežnika in povezavo naslovov s funkcijami,

ki jih napišemo v Pythonu. Ko brskalnik zahteva določen naslov, strežnik pokliče funkcijo, ki je povezana s tem naslovom in pošlje rezultat funkcije nazaj brskalniku.

Primer preproste spletne aplikacije:

```
1  from bottle import *
2
3  @route('/pozdravi')
4  def pozdrav():
5      return 'Pozdravljen, svet!'
6
7  run(host='localhost', port=8080, debug=True)
```

*@route()* je dekorator, ki poveže naslov s funkcijo. V tem primeru naslov */pozdravi* poveže s funkcijo *pozdrav()* (vrstici 3 in 4). Funkcija *run()* (vrstica 7) zažene vgrajen strežnik in čaka, da se brskalnik poveže z njim in mu pošlje zahtevo. Do strežnika dostopamo na naslovu *http://localhost:8080/*. Če bomo v brskalnik vnesli *http://localhost:8080/pozdravi*, se nam bo izpisala preprosta spletna stran z besedilom *Pozdravljen, svet!*.

Uporabljamo lahko tudi dinamične poti, ki ustrezajo več kot samo enemu naslovu. Dinamične poti sprejmejo niz znakov, ki ga lahko uporabimo kot argument funkcije. Da gre za dinamično pot, nakažemo z dvopičjem, ki mu sledi ime spremenljivke, v katero se shrani niz znakov. Primer:

```
1  @route('/pozdravi/:ime')
2  def pozdrav(ime):
3      return 'Zdravo, ' + ime + '!'
```

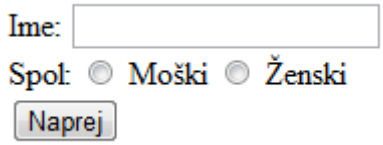
V tem primeru niz znakov, ki sledi */pozdravi/*, shranimo v spremenljivko *ime* in ga uporabimo v funkciji. Če bomo v spletni brskalnik vpisali */pozdravi/Janez*, se nam bo prikazala spletna stran s sporočilom *Zdravo, Janez!*. Podobno, če bomo vpisali */pozdravi/Micka*, se nam bo prikazala spletna stran s sporočilom *Zdravo, Micka!*. Če pa bomo vnesli le */pozdravi* ali pa */pozdravi/gospod/Novak*, nam bo strežnik vrnil sporočilo o napaki.

Privzeta HTTP metoda je GET. Če želimo uporabiti kakšno drugo metodo, npr. POST, moramo dekoratorju *route()* dodati *method='POST'*. Na istem naslovu lahko uporabimo različne metode, ki pokličejo različne funkcije.

Bottle omogoča dostop do podatkov v HTML POST obrazcih. Do podatka dostopamo z *request.forms.get('ime')*, kjer je *ime* ime, s katerim poimenujemo podatek. Primer:

```
1  @route('/pozdravi')
2  def pozdrav_vnos():
3      return '''<form method='POST' action='/pozdrav'>
4                  Ime: <input type='text' name='ime'/>
5                  Spol: <input type='radio' name='spol' value='m'/> šMoki
6                        <input type='radio' name='spol' value='z'/> Ženski
7                  <input type='submit' value='Naprej'>
8                 </form>'''
9
10 @route('/pozdravi', method='POST')
11 def pozdrav():
12     ime = request.forms.get('ime')
```

```
13      spol = request.forms.get('spol')
14      if spol == 'z':
15          return 'Pozdravljena, ' + ime '!'
16      else:
17          return 'Pozdravljen, ' + ime '!'
```




Slika 6. Obrazec, kamor uporabnik vpiše ime in spol.

V primeru je naslov */pozdravi* povezan z dvema funkcijama, ena ustreza metodi GET, druga pa metodi POST. Prva prikaže uporabniku obrazec, v katerega vpiše svoje ime in označi spol (Slika 6). Druga se sklicuje na vnesene podatke in na osnovi vnesenega prikaže spletno stran, ki vsebuje pozdrav. Do podatka v okencu za ime program dostopa z ukazom *request.forms.get('ime')*, do izbire spola pa z *request.forms.get('spol')*.

3.3. **Jinja2.** Da ne pišemo HTML kode neposredno v Python, saj ob bolj zapletenih spletnih straneh postane nepregledna, uporabljamo predloge. Jinja2 [8] je knjižnica za izdelavo HTML predlog, v katerih lahko uporabljamo Pythonovo programsko kodo.

3.3.1. *Pisanje predlog.* Predloge pišemo v HTML-ju, razširjenim s spremenljivkami in izrazi, ki jih ob prikazu nadomestijo vrednosti. Značka {% ... %} se uporablja za izvedbo stavkov, kot so zanke in pogojni stavki. Spremenljivke, ki jih predlogi pošlje aplikacija, zapišemo v dvojnih zavitih oklepajih {{...}}.

Primer predloge:

```
1  {% if len(uporabniki) > 0 %}
2  Uporabniki:
3    <ul>
4      {% for uporabnik in uporabniki %}
5        <li> {{uporabnik}} </li>
6      {% endfor %}
7    </ul>
8  {% else %}
9  Seznam uporabnikov je prazen.
10 {% endif %}
```

Vsak stavek mora imeti začetno in končno značko (vrstici 1 in 10 ter vrstici 4 in 6).

Če je v zgornjem primeru spremenljivka *uporabniki* seznam imen, na primer *uporabniki = ['Mojca', 'Janez', 'Micka']*, se bo izpisal seznam, ki bo imel tri elemente. Če je spremenljivka *uporabniki* prazen seznam, se bo izpisalo sporočilo *Seznam uporabnikov je prazen.*

3.3.2. *Uporaba predlog.* Predloge, ki smo jih napisali s pomočjo knjižnice Jinja2, v aplikaciji uporabimo na naslednji način:

```
1 from jinja2 import *
2
3 template = env.get_template('mojaPredloga.html')
4 return template.render(ime = 'Janez', priimek = 'Novak')
```

Z ukazom v vrstici 3 naložimo predlogo, katere ime podamo v oklepajih kot niz. V primeru je ime predloge *mojaPredloga.html*. Z ukazom v vrstici 4 predlogi pošljemo spremenljivke. V oklepaju naštejemo imena spremenljivk in njihove vrednosti. V primeru ima spremenljivka *ime* vrednost *'Janez'*, spremenljivka *priimek* pa vrednost *'Novak'*.

3.3.3. *Dedovanje predlog.* Ker morajo vse spletne strani vsebovati določene elemente in ker je večina spletnih strani podobnih, lahko uporabimo dedovanje predlog, da ne pišemo vsakič v predlogo vseh skupnih elementov. Dedovanje predlog nam omogoča, da naredimo ogrodje, ki vsebuje vse osnovne elemente spletne strani, in definiramo bloke, kamor podrejene predloge vstavijo lastno vsebino.

Primer ogrodja:

```
1  <!DOCTYPE html>
2  <html lang='si'>
3    <head>
4      <link rel='stylesheet' href='style.css'/>
5      <title>{% block naslov %} Naslov {% endblock %} </title>
6    </head>
7    <body>
8      {% block besedilo %} Tukaj vstavite besedilo {% endblock %}
9    </body>
10 </html>
```

V zgornjem ogrodju sestavimo osnovno strukturo spletne strani, naložimo stile in definiramo bloka *naslov* in *besedilo*.

Primer podrejene predloge:

```
1 {% extends 'base.html' %}
2 {% block besedilo %}
3      <h1> Vsebina strani </h1>
4      <p> To je vsebina strani. </p>
5 {% endblock %}
```

Na začetek podrejene predloge moramo obvezno napisati značko, ki pove, katero predlogo podrejena predloga razširja (vrstica 1). Če kakšnega bloka v podrejeni predlogi ne definiramo, se bo uporabila vsebina bloka iz ogrodja, sicer pa se celotna vsebina bloka prepiše. V podanem primeru bo v bloku *naslov* pisalo *Naslov*, saj v podrejeni predlogi ta blok ni definiran. V bloku *besedilo* bo namesto vsebine, ki je v ogrodju, zapisana vsebina, ki je v podrejeni predlogi.

3.4. **PuLP.** Linearne programe zapišemo in rešimo s pomočjo knjižnice PuLP [9]. Knjižnica linearnih programov ne rešuje sama, temveč ustvari pomožno datoteko in pokliče enega od nameščenih paketov za reševanje linearnih programov, v mojem primeru GLPK [10], ki dani program reši, nato pa vrne rešitev.

Primer uporabe knjižnice PuLP [11]:

```
1  from pulp import *
2  prob = LpProblem('test1', LpMinimize)
3  x = LpVariable('x', 0, 4)
4  y = LpVariable('y', -1, 1)
5  z = LpVariable('z', 0)
6  prob += x + 4*y + 9*z
7  prob += x + y <= 5
8  prob += x + z >= 10
9  prob += -y + z == 7
10 prob.solve()
11 for v in prob.variables():
12     print v.name, '=', v.varValue
13 print 'objective=', value(prob.objective)
```

Na začetku moramo definirati linearni program (vrstica 2). Funkcija *LpProblem* sprejme dva argumenta: ime problema in *LpMinimize*, če iščemo minimum, ali *LpMaximize*, če iščemo maksimum kriterijske funkcije. Nato moramo definirati spremenljivke, ki nastopajo v linearnem programu (vrstice 3 – 5). Funkcija *LpVariable* sprejme 4 argumente: ime spremenljivke, spodnjo mejo, zgornjo mejo in tip spremenljivke (*Integer* za celoštevilsko spremenljivko, *Binary* za 0/1 spremenljivko in *Continuous* za realno spremenljivko, kar je privzeta vrednost). Ko smo definirali spremenljivke, problemu dodamo kriterijsko funkcijo (vrstica 6), nato pa še omejitve (vrstice 7 – 9). Dani problem rešimo z ukazom *solve()* (vrstica 10), ki pokliče paket za reševanje linearnih programov. Do vrednosti kriterijske funkcije v optimalni rešitvi dostopamo z ukazom *value(prob.objective)*, kjer je *prob* ime linearnega programa, do vrednosti spremenljivk pa z *v.varValue*, kjer je *v* ime spremenljivke. Za izpis vseh spremenljivk lahko uporabimo naslednji ukaz:

```
1  for v in prob.variables():
2      print v.name, '=', v.varValue
```

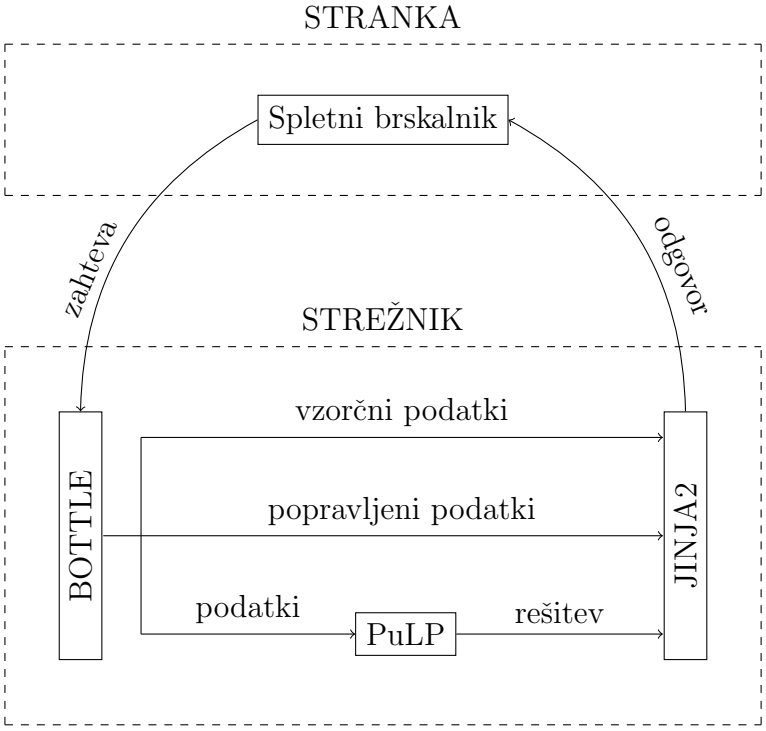
## 4. Spletna aplikacija

Delovanje spletne aplikacije je prikazano na sliki 7. Najprej brskalnik pošlje strežniku zahtevo po vzorčnem problemu, spremembi podatkov ali rešitvi problema. Na strežniku teče aplikacija (4.1) napisana v Pythonu s pomočjo knjižnice Bottle, ki pripravi odgovor na zahtevo. Če je bila zahtevana rešitev, jo izračuna s pomočjo knjižnice PuLP. Brskalniku pošlje odgovor v obliki HTML, ki ga naredi iz predloge napisane s pomočjo knjižnice Jinja2 (4.2).

4.1. **Program.** Program v Pythonu je sestavljen iz funkcij, ki so povezane s posameznimi naslovi, in iz pomožnih funkcij, ki so klicane znotraj le-teh.

```
1  @route('/')
2  def prvaStran():
3      template = env.get_template('zacetek.html')
4      return template.render()
```

Prva stran spletnega vmesnika (Slika 1) je enostavna spletna stran, ki se nahaja na naslovu /. Funkcija, ki pripada temu naslovu, pokliče predlogo *zacetek.html*, v kateri je zapisana vsebina prve strani.




Slika 7. Prikaz delovanja spletne aplikacije

```
1  @route('/vzorec/:tip', method = 'GET')
2  def funkcija(tip):
3      podatki = primeri[tip]
4      template = env.get_template(tip + '.html')
5      return template.render(podatki = podatki, tip = tip)
```

Vzorčni problemi (Slika 2) se nahajajo na dinamičnem naslovu */vzorec/:tip*, kjer je *tip* spremenljivka, ki pove, katerega izmed vzorčnih problemov je uporabnik izbral. Vsak vzorčni problem ima svoje podatke in predlogo. Na osnovi izbranega vzorčnega primera program izbere slovar z ustreznimi podatki, v katerem so imena spremenljivk in omejitev, matrika koeficientov, vektor relacij, vektor omejitev, koeficienti funkcije, ki jo optimiziramo, ter podatki, ali iščemo minimum ali maksimum te funkcije in ali naj bodo posamezne spremenljivke celoštevilske in/ali pozitivne. Aplikacija poišče ustrezno predlogo (vrstica 4) ter ji poda slovar s podatki in tip vzorca (vrstica 5). Slovar s podatki za naš primer je sledeč:

```
1  podatki = {'vrstice' : 8, 'stolpci': 5,
2             'spremenljivke': ['ovseni k.', 'koruzni k.', 'rozine', 'banana',
3                               'jogurt'],
4             'omejitve': ['mascobe', 'beljakovine', 'ogljikovi hidrati', 'vlaknine',
5                          'ovseni k.', 'koruzni k.', 'rozine', 'jogurt'],
6             'A': [[7.5, 1.6, 0, 0.3, 3.2], [8, 8.6, 1.1, 1.1, 3.5],
7                   [72.8, 85.1, 64.4, 19.9, 4.7], [7, 11, 6.8, 3.4, 0],
8                   [1, 0, 0, 0, 0], [0, 1, 0, 0, 0],
9                   [0, 0, 1, 0, 0], [0, 0, 0, 0, 1]],
10            ' relacije ': ['<=', '>=', '<=', '>=', '>=', '>=', '>=', '<='],
11            'b': [18, 12, 70, 7, 0.1, 0.1, 0.1, 2],
```

```
12              'optimum': 'min',
13              'c': [372, 368, 246, 79, 61],
14              'cele': [False, False, False, False, False],
15              'pozitivne': [True, True, True, True, True]
16              }
```

```
1  @route('/vzorec/:tip', method = 'POST')
2  def funkcija(tip):
3      podatki = preberi()
4      gumb = request.forms.get('gumb')
5      if gumb == 'Naprej':
6          status, funkcija, resitev = linearniProgram(podatki)
7          template = env.get_template('resitev.html')
8          return template.render(status = status, funkcija = funkcija,
9                                 resitev = resitev, podatki = podatki)
10     else:
11         if gumb == 'Dodaj omejitev':
12             podatki = dodajOmejitev(podatki)
13         elif gumb in ['remove{0}'.format(i) for i in range(podatki['vrstice'])]:
14             podatki = odstraniOmejitev(podatki, gumb)
15         elif gumb == 'Dodaj spremenljivko':
16             podatki = dodajSpremenljivko(podatki)
17         elif gumb in ['odstrani{0}'.format(j) for j in range(podatki['stolpci'])]:
18             podatki = odstraniSpremenljivko(podatki, gumb)
19         elif gumb == 'cela' or gumb == 'neCela':
20             podatki = cela(podatki, gumb)
21         elif gumb == 'pozitivna' or gumb == 'nePozitivna':
22             podatki = pozitivna(podatki, gumb)
23         template = env.get_template(tip + '.html')
24         return template.render(podatki = podatki, tip = tip).encode('utf-8')
```

Ko uporabnik na strani z vzorcem pritisne na enega od gumbov (Slika 3 in Slika 4), aplikacija najprej prebere vnesene podatke, saj se morajo vsakič vsi podatki na novo poslati na strežnik (vrstica 3, za več o funkciji *preberi* glej 4.1.1). Če je uporabnik izbral gumb *Naprej*, program pokliče funkcijo, ki zapiše in reši linearni program s pomočjo knjižnice PuLP (3.4) in vrne rešitev (vrstica 6, za več o funkciji *linearniProgram* glej 4.1.4). To rešitev aplikacija vstavi v ustrezno predlogo in jo pošlje spletnemu brskalniku (Slika 5). Če je uporabnik izbral katerega izmed gumbov za spreminjanje velikosti tabele, aplikacija pokliče funkcijo, ki ustrezno spremeni slovar s podatki in nato pošlje spletnemu brskalniku predlogo s spremenjeno tabelo (vrstica 24, za več o funkcijah glej 4.1.2 in 4.1.3).

4.1.1. *Funkcija preberi.* Ker strežnik nima spomina, mora aplikacija vsakič znova dobiti vse potrebne podatke.

```
1  def preberi():
2    form = request.forms
3    podatki = {}
4    podatki['A'] = [[float(form.get('a{0}{1}'.format(i, j)))
5      for j in range(stolpci)] for i in range(vrstice)]
6    ...
```

```
7     return podatki
```

Podan je primer, kako aplikacija prebere podatke, ki jih shrani v matriko $A$. V predlogi je vsako polje, ki predstavlja matriko $A$, poimenovano z *'aij'*, kjer je $i$ številka vrstice (omejitve), $j$ pa številka stolpca (spremenljivke), ki ji ta element pripada. Tako do podatka v 3. vrstici in 5. stolpcu dostopamo z ukazom *request.forms.get('a35')*. Na podoben način program dostopa do ostalih podatkov, ki jih shrani v slovar *podatki*.

4.1.2. *Funkcija dodajOmejitev.* Ko uporabnik želi dodati omejitev (vrstico v tabeli), aplikacija pokliče naslednjo funkcijo:

```
1  def dodajOmejitev(podatki):
2      podatki['vrstice'] += 1
3      podatki['A'] += [[0 for i in range(podatki['stolpci'])]]
4      podatki['omejitve'] += [None]
5      podatki['relacije'] += [None]
6      podatki['b'] += [0]
7      return podatki
```

Vsem elementom slovarja *podatki*, ki so odvisni od vrstic, funkcija na konec doda prazen element. Podobna je funkcija *dodajSpremenljivko*, ki jo aplikacija pokliče, če je uporabnik želel dodati spremenljivko (stolpec v tabeli).

4.1.3. *Funkcija odstraniSpremenljivko.* Če je uporabnik želel odstraniti določeno spremenljivko (stolpec v tabeli), aplikacija pokliče funkcijo *odstraniSpremenljivko*. Funkcija iz slovarja *podatki* vsem elementom, ki se nanašajo na stolpce, izbriše podatek, ki je pripadal izbranemu stolpcu.

```
1  def odstraniSpremenljivko(podatki, gumb):
2      podatki['stolpci'] -= 1
3      odstrani = int(gumb.strip('odstrani'))
4      del podatki['spremenljivke'][odstrani]
5      del podatki['c'][odstrani]
6      del podatki['cele'][odstrani]
7      del podatki['pozitivne'][odstrani]
8      for i in range(podatki['vrstice']):
9          del podatki['A'][i][odstrani]
10     return podatki
```

Kateri element je potrebno odstraniti, program prebere iz imena gumba, ki je sestavljen iz niza 'odstrani' in številke stolpca. Funkcija *odstraniOmejitev*, ki jo aplikacija pokliče, če je uporabnik želel odstraniti določeno omejitev (vrstico v tabeli), je podobna.

4.1.4. *Funkcija linearniProgram.* Ko uporabnik želi rešitev svojega problema, aplikacija pokliče funkcijo *linearniProgram*. Na osnovi vnesenih podatkov funkcija zapiše linearni program s pomočjo knjižnice PuLP (3.4) in ga reši.

```
1  def linearniProgram(podatki):
2    if podatki['optimum'] == 'min':
3      problem = LpProblem('Linearni program', LpMinimize)
4    else: problem = LpProblem('Linearni program', LpMaximize)
5    stolpci = podatki['stolpci']
```

```
6     sprem = []
7     for j, (cela, pozitivna) in
8      enumerate(zip(podatki['cele'], podatki['pozitivne'])):
9       sprem.append(LpVariable(podatki['spremenljivke'][j],
10        0 if pozitivna else None, cat = 'Integer' if cela else 'Continuous'))
11    problem += lpSum([podatki['c'][j] * sprem[j] for j in range(stolpci)])
12    for i in range(podatki['vrstice']):
13      if podatki['relacije'][i] == '>=':
14        problem += lpSum([podatki['A'][i][j] * sprem[j]
15          for j in range(stolpci)]) >= podatki['b'][i]
16      elif podatki['relacije'][i] == '=':
17        problem += lpSum([podatki['A'][i][j] * sprem[j]
18          for j in range(stolpci)]) == podatki['b'][i]
19      elif podatki['relacije'][i] == '<=':
20        problem += lpSum([podatki['A'][i][j] * sprem[j]
21          for j in range(stolpci)]) <= podatki['b'][i]
22    problem.solve()
23    status = LpStatus[problem.status]
24    resitev = {}
25    for sprem in problem.variables():
26      resitev[sprem.name] = sprem.varValue
27    funkcija = value(problem.objective)
28    return status, funkcija, resitev
```

4.2. **Predloge.** Za izdelavo HTML predlog sem uporabila knjižnico Jinja2 (3.3), za lepši izgled strani pa CSS stil Bootstrap [12]. Z dedovanjem predlog sem zagotovila enako strukturo vseh strani v spletnem vmesniku. Ogrodje spletne strani je videti takole:

```
1  <!DOCTYPE html>
2  <html lang='si'>
3    <head>
4      <meta charset='utf-8'>
5      <title>{% block title %}Naslov{% endblock %}</title>
6      <link href='/static/css/bootstrap.css' rel='stylesheet'>
7    </head>
8    <body>
9      <div class='container'>
10       <div class='row'>
11         <div class='span12'>
12           {% block besedilo %} {% endblock %}
13         </div>
14       </div>
15     </div>
16   </body>
17 </html>
```

Spletno stran sestavljata naslov (vrstica 5) in vsebinski del, kamor podrejene predloge vstavijo ustrezno vsebino (vrstica 12).

Predloga z vzorcem vsebuje opis vzorčnega problema, ki je del podrejene predloge, in tabelo, v katero uporabnik vnese podatke svojega problema. Tabela je del HTML obrazca, ki posreduje podatke na strežnik, in je vsem vzorčnim problemom skupna. Predloga iz aplikacije dobi podatke (slovar *podatki*), ki jih vstavi v tabelo.

```
1  <form action='{{tip}}' method='post'>
2    <input name='vrstice' type='hidden' value='{{podatki["vrstice"]}}'/>
3    <input name='stolpci' type='hidden' value='{{podatki["stolpci"]}}'/>
4    <table>
```

Za tiste podatke, ki jih želimo posredovati strežniku, uporabnik pa jih ne potrebuje, sem uporabila *input, type='hidden'*.

```
5  <thead>
6   <tr>
7    <th> Omejitve \ Spremenljivke </th>
8    {% for stolpec in range(podatki['stolpci']) %}
9     <th>
10     <input type='text' name='spremenljivka{{stolpec}}'
11       value='{{podatki["spremenljivke"][j]}}'/>
12     <button name='gumb' type='submit' value='odstrani{{j}}'/>
13    </th>
14   {% endfor %}
15   <th>
16    <button name='gumb' type='submit' value='Dodaj spremenljivko'/>
17   </th>
18  </tr>
19 </thead>
```

V glavi tabele so polja za vnos imen spremenljivk. Da je predloga uporabna za poljubno število spremenljivk, sem uporabila for zanko po spremenljivkah. Za vsako spremenljivko se ustvarita polje za vnos imena spremenljivke in gumb, s katerim lahko odstranimo spremenljivko iz tabele (vrstica 12). Vsako polje za vnos ima svoje ime, ki je sestavljeno iz niza 'spremenljivka' in zaporedne številke. Na osnovi tega imena aplikacija dostopa do vrednosti v polju. Vrednost, ki je zapisana v polju, predloga dobi iz slovarja *podatki*. Gumb za odstranitev spremenljivke je tipa 'submit'. Vsakič, ko uporabnik želi odstraniti spremenljivko, spletni brskalnik strežniku pošlje vse podatke, ki so zapisani v obrazcu. Na osnovi vrednosti, ki jo ima gumb, aplikacija določi, kateri gumb je bil izbran. Vrednost gumba za odstranitev spremenljivke je sestavljena iz niza 'odstrani' in zaporedne številke. Na koncu vrstice, ki je v glavi tabele, je še gumb, ki doda novo spremenljivko (vrstica 16).

Podobno je sestavljen ostali del tabele, kjer so polja za vnos preostalih podatkov.

4.3. **Izboljšave in omejitve.** Spletno aplikacijo bi se dalo še izboljšati. Večja izbira vzorčnih primerov, več dokumentacije o spletni aplikaciji in natančnejša navodila o uporabi vmesnika, bi naredili spletno aplikacijo lažjo za uporabo in bolj dostopno širšemu krogu uporabnikov. Ena izmed možnih izboljšav bi bila tudi ta, da bi namesto ročnega vnašanja lahko podatke naložili preko datoteke. S tem bi se izognili zamudnemu vpisovanju podatkov in možnim napakam, še posebej, če imamo podatke že v elektronski obliki. Prav tako bi se izognili hrošču v knjižnici Bottle, ki se pojavi pri uporabi več kot 100 polj v spletnem obrazcu.

## 5. Zaključek

V delu diplomskega seminarja sem v Pythonu naredila spletno aplikacijo za reševanje linearnih programov. Za izdelavo spletnega vmesnika sem uporabila Bottle, HTML predloge sem izdelala s pomočjo knjižnice Jinja2, za reševanje linearnih programov pa sem uporabila PuLP.

S spletno aplikacijo sem omogočila, da lahko uporabniki, ki nimajo potrebnega matematičnega znanja, sami rešujejo linearne optimizacijske probleme.

## Literatura


[1] *Web applications*, [ogled 10.1.2012], dostopno na `http://webtrends.about.com/od/webapplications/a/web_application.htm`

[2] *Tabele z energijsko in biološko vrednostjo živil*, [ogled 8.4.2012], dostopno na `www.zdravo-hujsam.si/pdf/Tabela_kaloricnih_vrednosti.doc`

[3] F. Hillier in G. Lieberman, *Introduction to operations research*, McGraw-Hill Higher Education, Boston, 2010.

[4] R. Vanderbei, *Linear programming: foundations and extensions*, Springer, New York, 2008.

[5] R. Bronson, *Shaum's outline of theory and problems of operations research*, Shaum's outline series, McGraw-Hill, New York, 1982.

[6] *HTTP Made Really Easy*, [ogled 8.4.2012], dostopno na `http://jmarshall.com/easy/http/`.

[7] *Bottle*, [ogled 8.4.2012], dostopno na `http://bottlepy.org/docs/dev/`.

[8] *Jinja*, [ogled 8.4.2012], dostopno na `http://jinja.pocoo.org/`.

[9] *PuLP*, [ogled 8.4.2012], dostopno na `http://packages.python.org/PuLP/`.

[10] *GLPK*, [ogled 10.8.2012], dostopno na `http://www.gnu.org/software/glpk/`.

[11] *PuLP*, [ogled 8.4.2012], dostopno na `http://pypi.python.org/pypi/PuLP/1.1/`

[12] *Bootstrap*, [ogled 8.4.2012], dostopno na `http://twitter.github.com/bootstrap/`.