SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA CHEMICKEJ A POTRAVINÁRSKEJ TECHNOLÓGIE

EVIDENČNÉ ČÍSLO: FCHPT-5415-70732

IMPLEMENTÁCIA OPTIMALIZAČNÝCH ALGORITMOV V JAVASCRIPTE

BAKALÁRSKA PRÁCA

2015 **Jakub Jakabšic**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA CHEMICKEJ A POTRAVINÁRSKEJ TECHNOLÓGIE

IMPLEMENTÁCIA OPTIMALIZAČNÝCH ALGORITMOV V JAVASCRIPTE

BAKALÁRSKA PRÁCA

FCHPT-5415-70732

Študijný program: B-AIM automatizácia, informatizácia a manažment v chémii a potravinárstve
Číslo študijného odboru: 2621
Názov študijného odboru: 5.2.14 automatizácia, 5.2.52 priemyselné inžinierstvo
Školiace pracovisko: Ústav informatizácie, automatizácie a matematiky (FCHPT)
Vedúci záverečnej práce: doc. Ing. Michal Kvasnica, PhD.

Bratislava 2015 **Jakub Jakabšic**

Slovenská technická univerzita v Bratislave
Ústav informatizácie, automatizácie a matematiky

Fakulta chemickej a potravinárskej technológie
Akademický rok: 2014/2015
Evidenčné číslo: FCHPT-5415-70732

STU FCHPT

ZADANIE BAKALÁRSKEJ PRÁCE

Študent:	**Jakub Jakabšic**
ID študenta:	70732
Študijný program:	automatizácia, informatizácia a manažment v chémii a potravinárstve
Kombinácia študijných odborov:	5.2.14. automatizácia, 5.2.52. priemyselné inžinierstvo
Vedúci práce:	doc. Ing. Michal Kvasnica, PhD.

Názov práce: **Implementácia optimalizačných algoritmov v Javascripte**

Špecifikácia zadania:

Cieľom práce je vytvoriť webovú platformu na riešenie optimalizačných problémov. Takéto riešenie sa bude skladať z dvoch hlavných komponentov:
* algoritmus na riešenie lineárneho programovania, implementovaný v jazyku Javascript
* sada webových stránok na komunikáciu užívateľa s algoritmom

Úlohy:
* v jazyku Javascript implementovať simplexovú metódu na riešenie lineárneho programovania
* vytvoriť formát vstupno/výstupných dát pre algoritmus
* vytvoriť webové stránky a formuláre na vkladanie údajov do algoritmu, zobrazenie výsledkov ako i vizualizáciu behu algoritmu v podobe zobrazenia iterácii simplexovej tabuľky

Rozsah práce: 40

Riešenie zadania práce od: 16. 02. 2015
Dátum odovzdania práce: 24. 05. 2015

L. S.

Jakub Jakabšic
študent

prof. Ing. Miroslav Fikar, DrSc.
vedúci pracoviska

prof. Ing. Miroslav Fikar, DrSc.
garant študijného programu

Poďakovanie

Chcel by som sa týmto poďakovať svojmu školiteľovi doc. Ing. Michalovi Kvasnicovi, PhD. za odbornú pomoc a usmernenie pri písaní mojej práce, za cenné rady, no hlavne za ochotu jasne a rýchlo odpovedať na všetky moje otázky.

Abstrakt

Cieľom tejto bakalárskej práce bolo vytvorenie programu (softwaru), ktorý sa zameriava na riešenie lineárnych optimalizačných problémov využitím Simplexovej metódy. Užívateľské prostredie je umiestnené do webového rozhrania, ktorého tvar je daný štruktúrou zdrojového dokumentu HTML a vzhľad upravený pomocou CSS.

Zdrojový kód programu je napísaný v programovacom jazyku JavaScript, ktorého úlohou je nielen vyriešenie daného problému, ale aj zobrazenie postupu riešenia pre užívateľa v prehľadnej forme.

V práci analyzujeme metódy a postupy, ktoré sme použili pri vývoji programu. V prvej časti práce sa venujeme teórii lineárneho programovania a opisu algoritmu metódy Simplex. V druhej časti práce sa venujeme hlavne opisu zdrojového kódu. Priblížime čitateľovi postup, akým zadávame do programu vstupy optimalizačného problému, ďalej postup, ktorým program zadaný problém vyrieši, a tiež analyzujeme spôsob, akým bude dané riešenie zobrazené pre užívateľa.

Práca má významný prínos pre zlepšovanie kvality výučby na fakulte, nakoľko tento software sprehľadňuje postup riešenia optimalizačných problémov a bude voľne dostupný pre študentov fakulty.

Kľúčové slová: lineárne programovanie, Simplexová metóda, HTML, JavaScript

Abstract

The main objective of the thesis was to develop a new program (software), which focuses on solving linear optimization problems using the Simplex method. User interface of this program was placed into the web environment, whose shape is determined by structure of the source HTML document, and its appearance adjusted by CSS.

The software´s source code is written in the programming language JavaScript. Its task is not only to solve the given problem, but also to display the solving procedure as a transparent output.

In the thesis, we analyze the methods and processes, which we´ve used for the program development. In the first section we focus on the theory behind linear programming and describe the Simplex algorithm. In the second section we mainly describe the program´s source code. We acquaint the reader with the procedure of how the input for an optimization problem is entered, then with the procedure of solving the given problem, and we also analyze the methods used to create the desired output appearance.

The thesis brings a significant contribution to improving the tuition quality at the faculty, as this software gives a transparent solving method for linear optimization and will be freely available for the faculty students.

Key words: Linear programming, Simplex method, HTML, JavaScript

Obsah

Zoznam ilustrácií

Zoznam skratiek a značiek

HTML – HyperText Markup Language – programovací jazyk na tvorbu web stránok, ktorého zdrojový kód, tzv. HTML dokument, pozostáva zo série elementov ohraničených tagmi v hranatých zátvorkách. Webový prehliadač tento dokument preloží a zobrazí elementy pre užívateľa.

CSS – Cascading Style Sheets – programovací jazyk na úpravu vzhľadu a iných vlastností HTML elementov

Slovník termínov

Tag je označenie kľúčového slova jazyka HTML, ktoré sa nachádza medzi ostrými zátvorkami. Vo väčšine prípadov má každý tag svoj pár, pričom prvý tag nazývame počiatočný a druhý ukončovací.

Element v HTML terminológii je označenie pre dvojicu spárovaných tagov, prípadne taký prvok dokumentu, ktorý zadávame iba jedným tagom, teda napr. obrázok alebo tlačidlo.

Obsah elementu je označenie pre text a elementy, ktoré sa nachádzajú medzi spárovanými tagmi.

Parameter elementu je označenie akejkoľvek nastaviteľnej vlastnosti elementu. Je nutné ho definovať v počiatočnom tagu. Príkladmi parametrov elementu sú: id, class, onClick...

Príklad elementu „p“: `<p id=“parameter_id“ >` *obsah elementu p* `</p>`

Parameter „id“ je špecifické označenie daného elementu. Id môže byť ľubovoľný text, avšak musí byť jedinečné v celom HTML dokumente.

Parameter „class“ je označenie pre skupinu elementov, ktoré majú byť CSS dokumentom upravené rovnakým spôsobom (prvky s rovnakým parametrom „class“ majú rovnakú veľkosť písma, farbu textu, pozadia a pod.).

Parameter „onClick“ – tento parameter je možné nastaviť pre všetky elementy, no je dôležitý hlavne pre element typu „button“, t.j. tlačidlo. Označuje akciu, ktorú má dokument (stránka) vykonať v momente, kedy užívateľ na dané tlačidlo klikne.

Element „div“ je označenie všeobecného blokového elementu v dokumente HTML. Vzhľad stránky sa upravuje hlavne využitím blokov „div“, nastavením ich parametrov.

Element „p“ označuje nový odstavec. Je to blokový element podobne ako „div“, s tým rozdielom, že automaticky posunie svoj obsah na nový riadok a urobí odsadenie pre lepšiu čitateľnosť.

Element „input“ je prázdne textové pole, do ktorého môže užívateľ stránky písať. Hlavným parametrom tohto elementu je „value“, ktorý označuje jeho obsah. Element input je definovaný iba jedným tagom.

Skript je označenie externého súboru vo formáte *.js* . Je to zdrojový kód programu napísaný v jazyku JavaScript, ktorý pozostáva z rôznych príkazov a funkcii, ktoré interagujú s elementmi HTML dokumentu (môžu meniť ich obsah aj parametre).

Cyklus „for“ je príkaz v jazyku JavaScript, ktorý ohraničuje sériu za sebou nasledujúcich príkazov, ktoré má program opakovane vykonať. Počet opakovaní cyklu „for“ je konečné číslo, definované pred začiatkom prvého opakovania.

Cyklus „while“ spĺňa rovnaký účel ako cyklus „for“, s tým rozdielom, že počet opakovaní nie je vopred známy. Na ukončenie cyklu slúži tzv. riadiaca podmienka – príkazy vo vnútri cyklu sa budú opakovať dovtedy, kým je riadiaca podmienka splnená.

Úvod

Dôvodov pre výber tejto témy bakalárskej práce bolo hneď viacero. Jedným z nich boli základné poznatky z programovania v jazyku JavaScript získané na cvičeniach z predmetu Informatizácia a informačné systémy.

Hlavným dôvodom pre výber tejto témy však bol predmet Optimalizácia, kde jedným z hlavných učebných okruhov bolo lineárne programovanie. V rámci neho sme sa naučili o Simplexovej metóde, jej histórii, logike, a princípe na akom funguje. Najväčšou zaujímavosťou tohto učiva sa stal fakt, že postup riešenia Simplexovou metódou zostáva rovnaký (a jednoduchý) aj pre veľmi rozsiahle optimalizačné problémy.

Keď sme však na cvičeniach počítali príklady z lineárnej optimalizácie využitím MATLAB-u, spôsob zadávania problému do softwaru bol pomerne zložitý a neprehľadný, dokonca aj pre veľmi malé optimalizačné problémy, a pritom je riešenie pomocou Simplexovej tabuľky také jednoduché.

Cieľom práce sa preto stalo vyriešenie tohto problému. Plán bol vytvoriť vlastný software na riešenie lineárnych optimalizačných problémov, ktorý by na riešenie využíval práve Simplexovú metódu, a vyžadoval by vstup v jednoduchšom formáte ako MATLAB, bol prehľadný a zároveň poučný.

Ideálnym prostredím na vytvorenie takéhoto programu sa stalo prostredie webovej stránky upravené JavaScriptom, pretože je doň možné implementovať algoritmus Simplexovej metódy, a zároveň spĺňa požiadavky na prehľadnejší vzhľad výstupu.

1 Ciele práce

Cieľom tejto práce je vytvoriť webovú stránku na serveri fakulty, ktorá bude prístupná pre internetovú verejnosť. Účel, za ktorým stránku vytvárame, je prevažne edukačný – je určená najmä pre študentov druhého ročníka študijného programu B-AIM, konkrétne študentov predmetu Optimalizácia.

Hlavným prvkom webu bude nástroj, ktorý bude schopný určiť optimálne riešenie matematicky formulovaného problému lineárneho programovania, zadaného užívateľom. Na vyriešenie tohto problému nepotrebuje žiaden prídavný software ani webový doplnok (tzv. add-on), čo výrazne uľahčuje jeho používanie. Všetka algoritmizácia a výpočty budú totiž implementované v programovacom jazyku JavaScript, ktorý priamo komunikuje s webovým prehliadačom a HTML štruktúrou stránky, ktorú môžeme ľubovoľne meniť aj po jej načítaní, a teda bude možné zadať vstupy a získať výstup na tej istej stránke.

Nakoľko prostredie webových stránok má vďaka JavaScriptu takmer neobmedzené možnosti na úpravu vzhľadu rozhrania, bude tento nástroj vhodný na vizualizáciu získaných výsledkov a hodnôt premenných v optime, ako aj lepšie pochopenie logických a matematických súvislostí, ktoré sú spojené s riešením daného problému, v porovnaní s MATLAB-om, s ktorým sa pracuje na cvičeniach.

Súčasťou webu budú tiež riešené aj neriešené príklady z Optimalizácie, ktoré umožnia užívateľom precvičiť si znalosti získané na prednáškach a cvičeniach z Optimalizácie. Svoje vedomosti si budú môcť zlepšovať aj v teoretickej sekcii stránky, kde budú uvedené učebné texty a odkazy na externé zdroje, ktoré ešte podrobnejšie vysvetľujú danú problematiku.

2 Úvod do lineárneho programovania

2.1 Optimalizácia

Pojmom **Optimalizácia** označujeme hľadanie najlepšieho možného riešenia problému. **Matematické programovanie** nám umožňuje transformovať reálne procesy na matematické modely, ktoré je potom možné riešiť využitím matematického aparátu. (Berežný, Kravecová, s. 10)

V praxi majú optimalizačné problémy spravidla ekonomický charakter, t.j. hľadáme také riešenie pre výrobu, distribúciu, alokáciu práce, nákup surovín a pod., ktoré prinesie najväčší zisk, resp. so sebou nesie najmenšie náklady.

2.1.1 Matematická formulácia optimalizačného problému:

Je daná funkcia f, ktorá každému číslu z množiny M reálnych čísel priradí práve jedno reálne číslo. Funkciu f nazveme **účelová funkcia**, množinu M budeme nazývať **množinou prípustných riešení.** Platí, že M je podmnožinou definičného oboru funkcie f.

$$f : M \to \mathbb{R} \tag{1}$$

Riešením optimalizačného problému – minimalizácie – je taký prvok $x^* \in M$ pre ktorý platí, že hodnota účelovej funkcie pre všetky ostatné prvky z množiny M je väčšia alebo rovnaká ako hodnota pre x^*:

$$\forall x \in M : f(x) \geq f(x^*) \tag{2}$$

Hlavným kritériom na porovnanie kvality dvoch riešení je teda hodnota účelovej funkcie. Avšak riešením optimalizačného problému je vektor x^*, ktorý predstavuje optimálne hodnoty premenných problému.

Všeobecný zápis optimalizačného problému má tvar:

$$\begin{aligned} J^* &= \min\ f(x) \\ &s.t.\ x \in M \end{aligned} \tag{3}$$

V tomto zápise jednotlivé symboly predstavujú:

- $f(x)$ – účelová funkcia, ktorej minimum chceme dosiahnuť
- x – vektor optimalizačných premenných
- M – množina prípustných riešení daná ohraničeniami problému
- J^* – minimálna číselná hodnota účelovej funkcie – kritérium na určenie kvality riešenia pre daný vektor x

Riešenie optimalizačného problému môžeme všeobecne zapísať v tvare:

$$x^* = \arg\min f(x) \quad s.t.\ x \in M \tag{4}$$

kde:

- arg min – funkcia **arg min** predstavuje argumenty v minime funkcie $f(x)$, t.j. vracia číselnú hodnotu argumentov, nie funkcie
- x^* je vektor hodnôt optimalizačných premenných v optime, t.j. požadované riešenie

2.1.2 Riešenie konvexných problémov

Pri riešení optimalizačného problému – minimalizácie – predpokladáme, že existuje také riešenie, ktoré by spĺňalo podmienku uvedenú v rovnici (2). Na splnenie tejto podmienky musí vo funkcii existovať globálne minimum na intervale, ktorý predstavuje množina M.

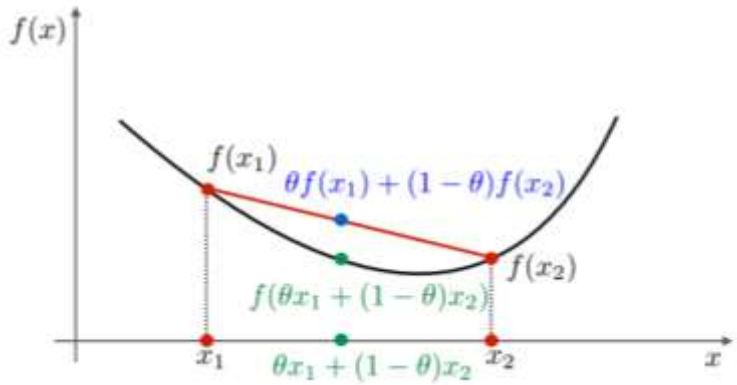
Spojitú funkciu nazývame **konvexnou** práve vtedy, keď všetky body funkcie na intervale (x_1, x_2) ležia nad spojnicou bodov $|f(x_1), f(x_2)|$, kde x_1 a x_2 predstavujú ľubovoľné dva body z definičného oboru funkcie. Presná matematická formulácia má tvar:

$$f(\theta x_1 + (1-\theta)x_2) \le \theta f(x_1) + (1-\theta) f(x_2) \tag{5}$$

Výraz $\theta x_1 + (1-\theta)x_2$ predstavuje ľubovoľný bod medzi bodmi x_1 a x_2. Ak platí, že funkčná hodnota takéhoto bodu je vždy menšia alebo rovná hodnote, ktorú v tomto bode nadobúda spojnica bodov x_1 a x_2, funkciu môžeme označiť ako konvexnú.

Na Obr. 1 vidíme jednorozmerný konvexný problém, pre ktorý zjavne existuje minimum.

Opakom konvexnej funkcie je **konkávna** funkcia (Pre definíciu konkávnej funkcie stačí v rovnici (5) otočiť znamienko nerovnosti). Vynásobením konvexnej funkcie číslom -1 teda dostaneme funkciu konkávnu a naopak.



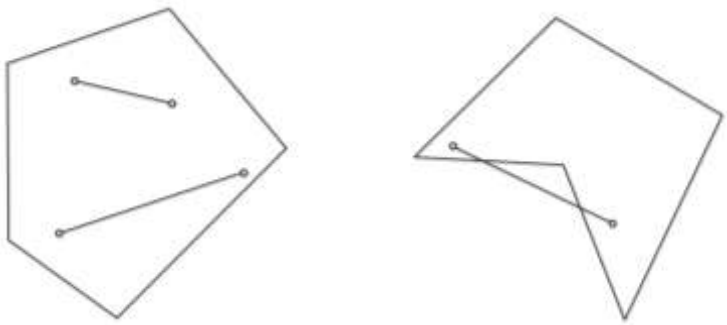
Obr. 1 - vlastnosť konvexnej funkcie

2.1.3 Definícia konvexnej množiny

Neprázdnu množinu $M \in \Re^n$ nazývame konvexnou, ak platí (Berežný, Kravecová, s. 32) :

$$(\forall x, y \in M)(\forall \lambda \in \langle 0;1 \rangle): \lambda x + (1-\lambda) y \in M \qquad (6)$$

Znamená to, že pre ľubovoľné dva body x a y, ktoré sú prvkami množiny M, sa musí každý bod ich spojnice takisto nachádzať v množine M. Konvexnosť a nekonvexnosť množiny môžeme ilustrovať v dvojrozmernom priestore, t.j. ak $M \subseteq \Re^2$:



Obr. 2 - príklad konvexnej a nekonvexnej množiny v $\mathbb{R}^2$

2.2 Lineárne programovanie

Lineárnym programovaním nazývame postup riešenia takého optimalizačného problému, ktorého účelová funkcia je lineárna, teda v tvare:

$$f(x_1, x_2, ..., x_n) = c_1 x_1 + c_2 x_2 + ... + c_n x_n \quad ; \quad c_{1...n} \in \Re \tag{7}$$

a ohraničenia problému tvorí sústava lineárnych rovníc a nerovníc.

Úlohu lineárneho programovania je možné zapísať v maticovom tvare:

$$\begin{aligned} &\min_x \quad c^T x \\ &s.t. \quad Ax \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} b \end{aligned} \tag{8}$$

kde jednotlivé symboly predstavujú:

- x je vektor optimalizačných premenných

$$x = (x_1 \quad x_2 \quad \cdots \quad x_n)^T \tag{9}$$

- c je vektor argumentov účelovej funkcie, ktorý má rovnaký rozmer ako vektor x. c^T predstavuje transponovaný vektor, ktorým je možné násobiť x.

$$c^T = (c_1 \quad c_2 \quad \cdots \quad c_n), \quad \forall i \in \langle 1, ..., n \rangle : \quad c_i \in \Re \tag{10}$$

- A predstavuje maticu koeficientov ohraničení, ktoré vymedzujú množinu prípustných riešení. Počet ohraničení, t.j. počet riadkov matice označíme m

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}; \quad \begin{matrix} \forall i \in \langle 1, ..., m \rangle \\ \forall j \in \langle 1, ..., n \rangle \end{matrix} : \quad a_{ij} \in \Re \tag{11}$$

- b je vektor pravých strán ohraničení. Dva vektory - $(A \cdot x)$ a b je možné porovnať práve vtedy, keď sa rovnajú ich rozmery. Preto:

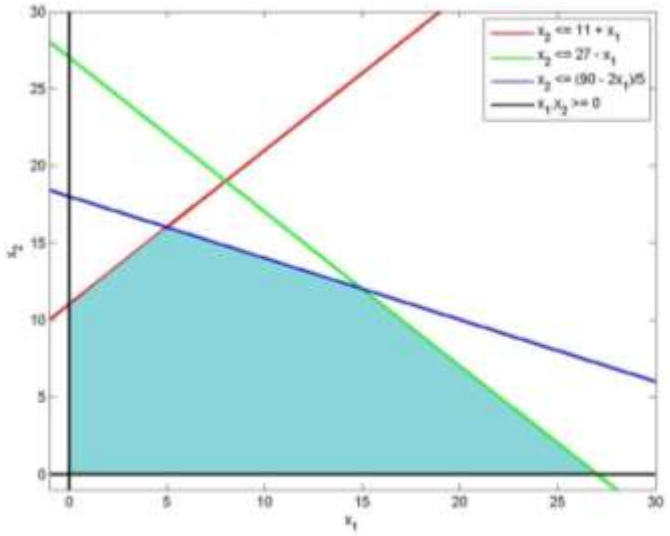
$$b = (b_1 \quad b_2 \quad \cdots \quad b_m)^T, \quad \forall i \in \langle 1, ..., m \rangle : \quad c_i \in \Re \tag{12}$$

2.2.1 Grafická interpretácia problémov lineárneho programovania

Majme dvojrozmerný lineárny optimalizačný problém, ktorý je zadaný v tvare:

$$\begin{aligned} \max \quad & 4x_1 + 6x_2 \\ s.t. \quad & -x_1 + x_2 \le 11 \\ & x_1 + x_2 \le 27 \\ & 2x_1 + 5x_2 \le 90 \\ & x_1, x_2 \ge 0 \end{aligned} \tag{13}$$

Vidíme, že všetky zadané ohraničenia sú v tvare nerovnosti. V karteziánskej súradnicovej sústave danej premennými x_1 a x_2 ($x_2 = f(x_1)$) predstavujú tieto ohraničenia polroviny (ohraničenia v tvare rovnosti by predstavovali priamky). Množina prípustných riešení M je daná prienikom všetkých ohraničení. Všetky ohraničenia zakreslíme do sústavy a získame tak množinu M .



Obr. 3 - grafická interpretácia množiny prípustných riešení M

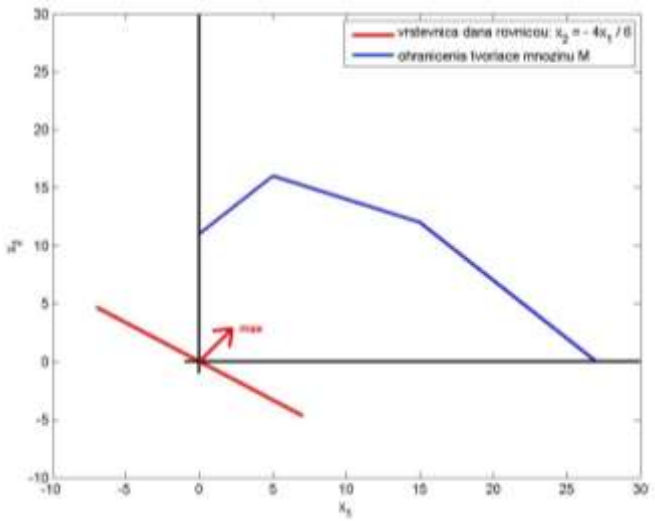
Účelová funkcia je daná rovnicou

$$4x_1 + 6x_2 = J \quad , \quad J \in \Re \tag{14}$$

Rovnica priamky v súradnicovej sústave $x_2 = f(x_1)$, daná účelovou funkciou má tvar:

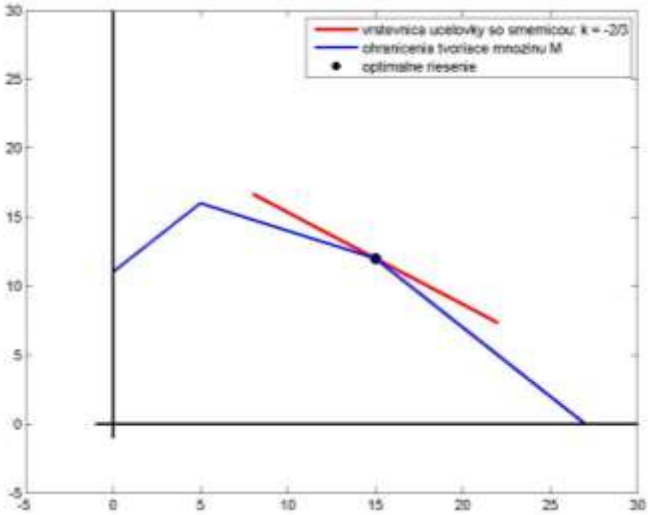
$$x_2 = -\frac{2}{3}x_1 + \frac{J}{6} \tag{15}$$

Vidíme, že smernica tejto priamky sa s hodnotou účelovej funkcie J nemení. Nazveme ju preto **vrstevnica účelovej funkcie**. Položme najprv $J = 0$ a zakreslime vrstevnicu do sústavy:



Obr. 4 - zakreslenie vrstevnice do grafu

Cieľom riešenia je maximalizovať hodnotu J, pričom zvyšovaním tejto hodnoty sa vrstevnica posúva smerom nahor (zvyšovaním J rastie x_2). Hľadáme preto najvzdialenejší bod množiny prípustných riešení M v smere rastu J (smer vyznačený šípkou).



Obr. 5 - grafické znázornenie optimálneho riešenia

Posúvaním vrstevnice smerom nahor sme našli bod, ktorý je súčasťou množiny M a pre každý iný bod z M má účelová funkcia menšiu hodnotu. Tento bod je hľadaným optimálnym riešením problému.

Optimálne riešenie je teda dané prienikom dvoch priamok ohraničení:

$$\begin{aligned} &I. \quad x_2 = -x_1 + 27 \\ &II. \quad x_2 = -\frac{2}{5}x_1 + 18 \end{aligned} \tag{16}$$

Hodnoty premenných v optime dostaneme vyriešením tejto sústavy rovníc:

$$\begin{aligned} x_1^* &= 15 \\ x_2^* &= 12 \end{aligned} \tag{17}$$

Dosadením x_1^* a x_2^* do účelovej funkcie zistíme jej optimálnu hodnotu:

$$J^* = 132 \tag{18}$$

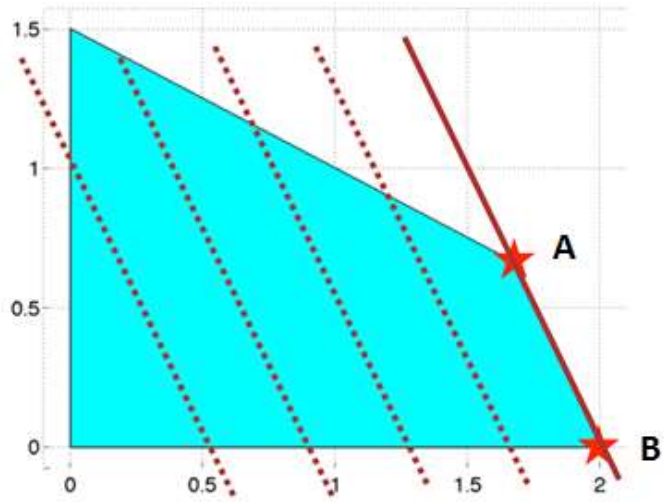
2.2.2 Viacnásobné optimá

Môže nastať situácia, že vrstevnica účelovej funkcie je rovnobežná s hranicou množiny prípustných riešení, ktorá je najďalej v smere zvyšovania hodnoty účelovej funkcie pri maximalizácii, resp. v smere znižovania pri minimalizácii. Inými slovami, priamka účelovej funkcie a priamka aktívneho ohraničenia problému majú rovnakú smernicu.

Príklad:

$$\begin{aligned} \max \quad & x_1 + 0.5x_2 \\ & 2x_1 + x_2 \le 4 \\ s.t. \quad & x_1 + 2x_2 \le 3 \\ & x_1, x_2 \ge 0 \end{aligned} \tag{19}$$

Do grafu zakreslíme množinu prípustných riešení, vrstevnicu účelovej funkcie a posúvame ju smerom k zlepšovaniu riešenia.



Obr. 6 - grafické znázornenie viacnásobného optima

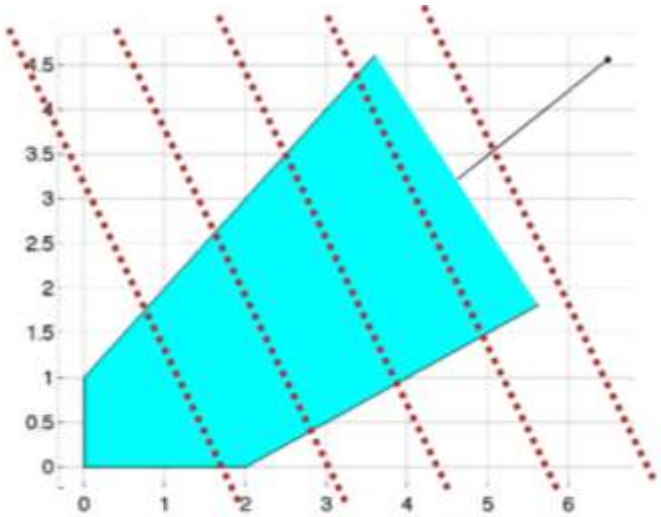
Nakoľko je priamka prvého (aktívneho) ohraničenia rovnobežná s vrstevnicou, optimálnym riešením je v takomto prípade ktorýkoľvek bod ležiaci na úsečke AB – hodnota účelovej funkcie je pre všetky body úsečky rovnaká a teda sú všetky riešenia ekvivalentné.

2.2.3 Neohraničené optimum

Doteraz sme uvažovali, že množina prípustných riešení M je konvexná a uzavretá. Môže však nastať situácia, že množina M nie je ohraničená v smere zvyšovania hodnoty účelovej funkcie.

Príklad:

$$\begin{aligned} \max \quad & 2x_1 + x_2 \\ & -x_1 + x_2 \le 1 \\ s.t. \quad & x_1 - 2x_2 \le 2 \\ & x_1, x_2 \ge 0 \end{aligned} \tag{20}$$



Obr. 7 - neohraničená množina prístupných riešení

Optimálna hodnota účelovej funkcie takto formulovaného problému je $\pm\infty$ podľa toho, či sa jedná o minimalizáciu alebo maximalizáciu, nepoznáme však optimálne hodnoty premenných. Znamená to, že úloha je zle naformulovaná a nemá riešenie.

2.2.4 Neriešiteľnosť

V prípade, že ohraničenia problému nemajú prienik ($M = \{\ \}$), neexistuje bod, ktorý by vyhovoval všetkým ohraničeniam. Riešením takto formulovaného problému je prázdna množina.

2.3 Štandardný tvar úloh lineárneho programovania

Hovoríme, že úloha lineárneho programovania je v **kanonickom tvare**, ak je zadaná v tvare:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} (c_i \; x_i) \\ s.t. \quad & \sum_{i=1}^{n} (a_{ji} \; x_i) \geq b_j \,, \quad pre \;\; j = 1, \ldots, m \\ & x_i \geq 0, \qquad pre \;\; i = 1, \ldots, n \end{aligned} \tag{21}$$

Štandardným tvarom úlohy lineárneho programovania s n premennými a m ohraničeniami nazývame tvar:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} (c_i \; x_i) \\ s.t. \quad & \sum_{i=1}^{n} (a_{ji} \; x_i) = b_j \,, \quad pre \;\; j = 1, \ldots, m \\ & x_i \geq 0, \qquad pre \;\; i = 1, \ldots, n \end{aligned} \tag{22}$$

maticový zápis štandardného tvaru má tvar:

$$\begin{aligned} \max \quad & c^T x \\ s.t. \quad & Ax = b \\ & x_i \geq 0, \quad pre \;\; i = 1, \ldots, n \end{aligned} \tag{23}$$

Všeobecný, kanonický a štandardný tvar úlohy lineárneho programovania sú navzájom ekvivalentné. Preto je možné transformovať akýkoľvek problém lineárneho programovania do štandardného tvaru. (Berežný, Kravecová, s. 32)

2.3.1 Prevod úlohy lineárneho programovania do štandardného tvaru

Transformácia problému do štandardného tvaru pozostáva z viacerých krokov:

1. Zmena extremalizácie účelovej funkcie

   Cieľom je zmeniť úlohu minimalizácie na maximalizáciu. Maximalizácia zápornej hodnoty premennej smerom k záporným číslam je ekvivalent minimalizácie danej premennej. Stačí preto vynásobiť účelovú funkciu číslom -1 :

$$\max -x \quad \Leftrightarrow \quad \min x \tag{24}$$

2. Transformácia nerovností na rovnosti

   Ohraničenia v tvare nerovnosti je možné transformovať na rovnosti pridaním doplnkových optimalizačných premenných, tzv. **slackov**. Slack predstavuje rozdiel medzi hodnotou ľavej strany nerovnice a hranicou danou pravou stranou:

$$\begin{aligned} \sum_{i=1}^{n}\left(a_{ji}\,x_i\right) \geq b_i \quad &\rightarrow \quad \sum_{i=1}^{n}\left(a_{ji}\,x_i\right) - s_i = b_i; \quad s_i \geq 0 \\ \sum_{i=1}^{n}\left(a_{ji}\,x_i\right) \leq b_i \quad &\rightarrow \quad \sum_{i=1}^{n}\left(a_{ji}\,x_i\right) + s_i = b_i; \quad s_i \geq 0 \end{aligned} \tag{25}$$

   Nakoľko tento rozdiel nemôže byť záporný, pridáme k ohraničeniam podmienku nezápornosti všetkých slackov.

3. Transformácia neohraničenej premennej na ohraničenú

   Pri riešení tejto úlohy vychádzame z faktu, že akékoľvek reálne číslo je možné zapísať ako rozdiel dvoch nezáporných čísel:

$$x_i \ \textit{je neohraničené} \quad \rightarrow \quad \begin{aligned} x_i &= x_i^{+} - x_i^{-} \\ x_i^{+} &\geq 0 \\ x_i^{-} &\geq 0 \end{aligned} \tag{26}$$

2.3.2 Maticový zápis prevodu problému do štandardného tvaru

Majme optimalizačný problém zadaný vo všeobecnom tvare.

$$\begin{aligned} &\max \quad c^T x \\ &s.t. \quad Ax \le b \end{aligned} \tag{27}$$

Prevod takto zapísaného problému na štandardný tvar pozostáva z nasledujúcich krokov:

1. Maximalizácia vyhovuje štandardnému tvaru, preto nie je potrebné meniť vektor argumentov účelovej funkcie c
2. Optimalizačné premenné (vektor x) nie sú ohraničené, štandardný tvar vyžaduje nezápornosť premenných. Spravíme preto substitúciu:

$$\begin{aligned} x &= x^+ - x^- \\ x^+ &\ge 0 \\ x^- &\ge 0 \end{aligned} \tag{28}$$

3. Prehodíme ohraničenia do tvaru rovnosti pridaním slackov :

$$\begin{aligned} Ax \le b \quad \rightarrow \quad & Ax + Is = b \\ & s \ge 0 \end{aligned} \tag{29}$$

4. Zlúčime vektory optimalizačných premenných x^+, x^-, s do jedného vektora z. Vzhľadom na to musíme upraviť aj vektory násobiace optimalizačné premenné v ohraničeniach a v účelovej funkcii. Výsledný tvar problému v štandardnom tvare:

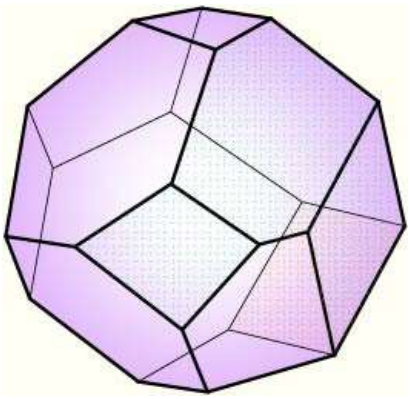
$$\begin{aligned} &\max \quad d^T z \\ &s.t. \quad Fz = b \\ &\qquad z \ge 0 \end{aligned} \quad \Leftrightarrow \quad \begin{aligned} &\max \quad \begin{bmatrix} c^T & -c^T & 0 \end{bmatrix} \cdot \begin{bmatrix} x^+ \\ x^- \\ s \end{bmatrix} \\ &s.t. \\ &\quad \begin{bmatrix} A & -A & I \end{bmatrix} \cdot \begin{bmatrix} x^+ \\ x^- \\ s \end{bmatrix} = b \\ &\quad z \ge 0 \end{aligned} \tag{30}$$

3 Simplexová metóda

Simplex algoritmus, alebo Simplexová metóda, je veľmi rozšírený spôsob riešenia úloh lineárneho programovania. Aby sme mohli aplikovať Simplex na vyriešenie problému, musí byť tento problém zadaný v štandardnom tvare, teda:

$$\begin{aligned} \max \quad & d^T z \\ s.t. \quad & Fz = b \\ & z \geq 0 \end{aligned} \tag{31}$$

V kapitole 1.2.1 sme opísali, ako z ohraničení v tvare nerovnosti vznikne konvexná množina prípustných riešení, ktorú sme označili *M*. Pre viacrozmerné problémy lineárneho programovania vytvárajú ohraničenia tzv. **konvexný mnohosten**.



Obr. 8 - príklad konvexného mnohostena

Bod $x = (x_1, \ldots, x_n)$ je vrcholom mnohostena práve vtedy a len vtedy, ak prvky stĺpcových vektorov A_i (A_i je podmnožina množiny všetkých ohraničení), ktorým prislúcha vektor nenulových vstupov x ($x_i \neq 0$) sú lineárne nezávislé. (Murty, s. 140). Vrcholy mnohostena teda predstavujú najmenšie možné priesečníky ohraničení. Takýto vrchol nazývame **zlučiteľné bázické riešenie**.

Možno dokázať, že ak hodnota účelovej funkcie *J* problému lineárneho programovania v štandardnom tvare nadobúda maximum v množine prípustných riešení *M*, nadobúda ho v aspoň jednom vrchole mnohostena (množiny *M*). (Murty, s. 141)

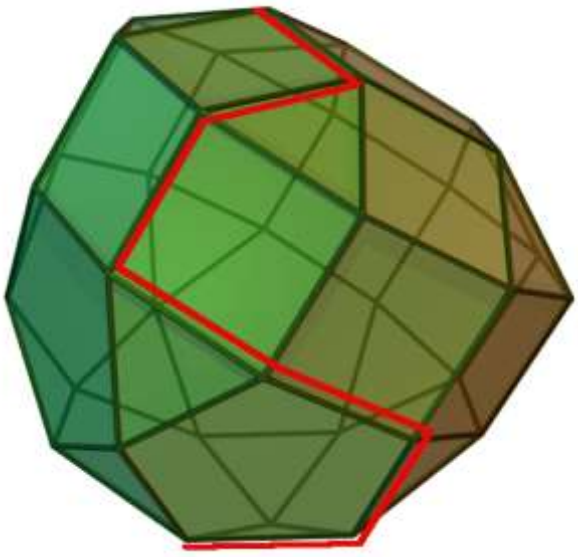
Optimálne riešenie problému sa teda musí nachádzať na jednom z vrcholov, čo znamená, že pri hľadaní tohto riešenia hľadáme jedno z konečného počtu možných riešení. Počet riešení teda nie je nekonečný, no napriek tomu je pre bežné optimalizačné problémy veľmi veľký. (Murty, s. 143)

Dá sa tiež dokázať, že ak vo vrchole x množiny M nenadobúda účelová funkcia maximum, t.j. ak x nie je optimálnym riešením, musí tento bod ležať na takej hrane mnohostena, po ktorej keď sa posúvame smerom od bodu x, hodnota účelovej funkcie sa buď zvyšuje, alebo nemení. (Murty, s.137)

Ak je táto spojnica nekonečná, jedná sa o typ optimalizačného problému, ktorý sme opísali v časti 1.2.3, teda neohraničený problém.

Ak je spojnica ukončená nejakým vrcholom y, tak platí že $f(y) \geq f(x)$.

Simplexová metóda využíva práve tieto poznatky. Prvým krokom pri riešení úlohy lineárneho programovania simplexovou metódou je nájsť nejaký počiatočný vrchol množiny prípustných riešení – zlučiteľné bázické riešenie, ktoré označíme x_0. Bod x_0 je prvkom viacerých spojníc. Treba vybrať takú spojnicu, na ktorej bude hodnota účelovej funkcie väčšia alebo rovná $f(x_0)$. Táto spojnica končí vo vrchole, ktorý označíme x_1. V prípade, že hodnota $f(x_1)$ nie je optimálna, znova hľadáme spojnicu, na ktorej je hodnota vyššia, a jej druhým vrcholom je x_2. Ak je problém ohraničený, týmto postupom sa dopracujeme k optimálnemu riešeniu.



Obr. 9 - ilustrácia Simplex algoritmu

3.1 Simplexová tabuľka

Majme optimalizačný problém, ktorý sme uviedli v kapitole 1.2.1 (rovnica (13)). Preveďme ho do štandardného tvaru:

$$\begin{aligned} \max \quad & 4x_1 + 6x_2 \\ s.t. \quad & -x_1 + x_2 + s_1 = 11 \\ & x_1 + x_2 + s_2 = 27 \\ & 2x_1 + 5x_2 + s_3 = 90 \\ & x_1, x_2, s_1, s_2, s_3 \geq 0 \end{aligned} \tag{32}$$

Problém si prepíšeme do tabuľky, ktorej jednotlivé riadky predstavujú argumenty všetkých optimalizačných premenných problému (vrátane slackov). Pridáme ešte jeden riadok na spodok tabuľky, ktorý bude obsahovať argumenty účelovej funkcie vynásobené číslom -1. Každý stĺpec tabuľky teda predstavuje jednu premennú. Výrazom **bázická premenná** označujeme takú premennú, ktorej stĺpec obsahuje práve jedno číslo 1 a zvyšok stĺpca je naplnený nulami.

x_1	x_2	s_1	s_2	s_3	PS
-1	1	1	0	0	11
1	1	0	1	0	27
2	5	0	0	1	90
-4	-6	0	0	0	0

Simplexová tabuľka 1

Prvým krokom je nájsť zlučiteľné bázické riešenie. Hodnota nebázických premenných bude 0, hodnota bázických premenných bude pravá strana (posledný stĺpec) toho riadku, v ktorom má daná bázická premenná jednotku.

$$\begin{aligned} & x_1 = 0 \quad x_2 = 0 \\ & s_1 = 11 \quad s_2 = 27 \quad s_3 = 90 \\ & J = 0 \end{aligned} \tag{33}$$

Simplex algoritmus je iteratívny a z každého zlučiteľného riešenia postupujeme rovnakým spôsobom:

- Nájdeme premennú, ktorej zvýšením sa najrýchlejšie zvýši hodnota účelovej funkcie., t.j. hľadáme takú premennú, ktorej argument v účelovej funkcii je najväčší - má v poslednom riadku najmenšie záporné číslo. Takto sme vybrali spojnicu, po ktorej sa posúvame smerom k optimálnemu riešeniu.
- Treba zistiť, ktoré ohraničenie ako prvé pretína túto spojnicu a vytvára tak vrchol, ktorý bude zlučiteľným bázickým riešením v ďalšom kroku. Hľadáme preto riadok (ohraničenie), v ktorom je pomer pravej strany a čísla v danom stĺpci najmenší.
- Keďže pre hodnotu účelovej funkcie je najvýhodnejšie zvyšovať hodnotu tejto premennej, ohraničenie nájdené v druhom kroku sa stane aktívnym, pričom zvyšovať budeme iba hodnotu tejto premennej. Jej hodnotu však nemôžeme zvýšiť viac, než dovoľuje toto ohraničenie. Z danej premennej teda vyrobíme bázickú, s jednotkou v riadku daného ohraničenia.
- Nasledujú elementárne riadkové operácie, pomocou ktorých vyrobíme z daného stĺpca bázický stĺpec. Tým sa upravia argumenty pre ostatné premenné vo všetkých ohraničeniach a aj v účelovej funkcii.
- Určíme hodnoty premenných pre nové zlučiteľné bázické riešenie.
- Zistíme, či je toto riešenie optimálne a ak nie je, pokračujeme znova prvým krokom

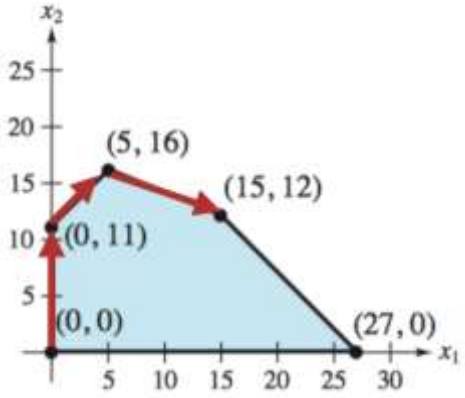
x_1	x_2	s_1	s_2	s_3	PS
-1	1	1	0	0	11
2	0	-1	1	0	16
7	0	-5	0	1	35
-10	0	6	0	0	66

Simplexová tabuľka 2

Zlučiteľné bázické riešenie 2:

$$\begin{aligned} &x_1 = 0 \quad x_2 = 11 \\ &s_1 = 0 \quad s_2 = 16 \quad s_3 = 35 \\ &J = 66 \end{aligned} \tag{34}$$

Pokračujeme prvým bodom uvedeného postupu, a dopracujeme sa tak k optimálnemu riešeniu. Graficky môžeme postup riešenia tohto problému ilustrovať takto:



Obr. 10 - iterácie smerom k optimálnemu riešeniu

Postup riešenia úloh Simplex metódou je bližšie popísaný v kapitole 4, kde sa priamo venujeme jej algoritmizácii.

4 Výsledky práce

4.1 Tvorba HTML štruktúry stránky

V súčasnej dobe existuje mnoho spôsobov na vytvorenie obyčajnej webovej stránky – použitím open-source platforiem ako napr. Drupal, ktorý funguje na základe jazyka PHP, alebo priamo pomocou PHP. Avšak jednou z často využívaných metód na tvorbu stránky je klasická HTML štruktúra, ktorej vzhľad upravujeme pomocou jazyka CSS. Pre účely našej stránky bude najvhodnejšia táto metóda, keďže vo výpočte a v konečnom zobrazení zohráva veľkú úlohu JavaScript, ktorý má veľkú synergiu s HTML a CSS.

Vytvorenie webovej stránky z celkom prázdneho zdrojového kódu nie je ľahká úloha a tak sme využili nástroje na internete. Základom HTML štruktúry našej stránky sa stala voľne dostupná HTML šablóna, ktorú sme získali zo stránky www.opendesigns.org. Využitie šablóny prináša viacero výhod oproti vytváraniu celej štruktúry:

- dokončená HTML štruktúra s „class“ parametrami
- vyhotovený CSS externý dokument, ktorý upravuje vzhľad jednotlivých častí stránky pomocou „class“ parametrov
- dokončené pozadie, interakcia menu s myšou, rôzne vizuálne prvky na stránke, ktoré fungujú samostatne

Použitie a iba dodatočná úprava šablóny nám ušetrilo veľké množstvo času, ktoré by sme museli vynaložiť na vytvorenie dizajnu stránky, ktorý je veľmi dôležitý pre celkový dojem užívateľa stránky, no napriek tomu nie je predmetom práce.

4.1.1 Vzhľad stránky

Vybranú šablónu bolo potrebné dodatočne upraviť, aby vyhovovala požiadavkám na stránku. Stránka je členená na tri časti: nadpis, ľavé menu a obsah stránky.

Nadpis nesie názov Optimalizácia Simplex metódou. Zostáva vždy rovnaký, a na rovnakej pozícii vzhľadom na plochu zobrazenú vo webovom prehliadači, pri vykonaní akejkoľvek akcie na stránke. Toto sme dosiahli tak, že všetky elementy nadpisu sú uložené v rámci jedného bloku `<div>`, ktorého css parameter `position` je nastavený na `fixed`.

Ľavé menu obsahuje odkazy na pohyb po jednotlivých sekciách stránky. V malých blokoch pod ním sú umiestnené externé odkazy, spolu so stručným popisom stránky na ktorú odkazujú.

Obsah stránky sa nachádza vpravo vedľa menu. Je to jediná dynamická časť stránky. Užívateľ si v menu vyberie jednu zo sekcii.

4.2 Sekcie stránky

Kliknutím na príslušný odkaz v menu sa dostávame k jednotlivým sekciám stránky.

4.2.1 Úvod

V úvode sa nachádzajú základné informácie o stránke: na čo slúži, prečo vznikla a tiež popisy jednotlivých sekcií stránky, ktorých úlohou je poučiť užívateľa o tom ako ďalej pokračovať.

4.2.2 Optimalizuj

Časť Optimalizuj je najdôležitejšou súčasťou stránky, keďže práve tu sa nachádza program na lineárnu optimalizáciu. Tejto sekcii sa preto venujeme v kapitole 5.3 – Jadro programu.

4.2.3 Metóda Simplex

Táto sekcia je určená výhradne pre študentov Optimalizácie. Nachádza sa tu postup riešenia lineárnych problémov a tiež učebné materiály k lineárnemu programovaniu. Študenti si teda môžu zopakovať učivo prebrané na prednáškach z iného zdroja, pričom si na tej istej stránke môžu tiež vyskúšať, čo sa práve naučili.

4.2.4 Príklady

V tejto sekcii sa nachádza zbierka riešených a neriešených príkladov. Riešené príklady ilustrujú správny postup riešenia optimalizačného problému využitím softwaru stránky. Neriešené príklady slúžia na precvičenie si matematickej formulácie optimalizačných problémov. Vďaka tomu, že programu stačí na vyriešenie problému jeho zadanie v štandardnom tvare, správnosť formulácie problému si môže užívateľ overiť veľmi rýchlo.

4.3 Jadro programu

Jadro programu sa nachádza v sekcii, ku ktorej sa dostaneme kliknutím na odkaz v ľavom menu „Optimalizuj“. V zdrojovom kóde HTML štruktúry, ktorý vytvára stránku *Optimalizácia* sa za ukončovacím tagom `</body>` nachádza tag `<script>`, ktorý obsahuje odkaz na externý súbor vo formáte *.js* . Dátový typ *.js* označuje súbor v jazyku JavaScript. Po načítaní všetkých HTML elementov stránky a ich úprave pomocou CSS sa pomocou tagu `<script>` načíta daný súbor, ktorý dynamicky upravuje vzhľad stránky.

4.3.1 Úvodný vzhľad stránky Optimalizácia

Obr. 11 - úvodný vzhľad stránky Optimalizácia

Na (Obr. 8) vidíme, že HTML štruktúra stránky je pomerne jednoduchá. Skladá sa z troch hlavných `<div>` blokov.

Prvý blok je vyplnený už v pôvodnom .html dokumente. Obsahuje nadpis „Optimalizácia“ a pod ním tabuľku bez viditeľných ohraničení. Jednotlivé bunky tabuľky obsahujú text, ktorý vidíme na obrázku. Použitie tabuľky nám zabezpečí požadovaný vizuálny efekt obsahu.

Počet opt. premenných	*id = pocet_premennych*	INPUT
Počet ohraničení	*id = pocet_ohraniceni*	

Obr. 12 - ilustrácia tabuľky v prvom bloku stránky Optimalizácia

Vytvorená tabuľka má dva riadky a tri stĺpce. V jazyku HTML je šírka stĺpca nastaviteľná priamo v `<td>` tagu bunky (parameter `width`), ktorá je súčasťou tohto stĺpca - nastavenie sa potom prejaví

pre všetky riadky stĺpca. Nastavíme teda šírku na hodnotu, ktorá zodpovedá požadovanej dĺžke textu, ktorý bude táto bunka obsahovať.

Do buniek druhého stĺpca sme vložili prvky `<input>`, pričom každému z nich sme priradili vlastný parameter `id`. V .css dokumente sme vytvorili novú triedu formátovania (*class*) pre tieto inputy, čím sme zabezpečili požadovanú veľkosť a zarovnanie textu v inputoch.

Tretí stĺpec tvorí len jedna zlúčená bunka. Tento stav sme dosiahli využitím parametra `rowspan`, ktorého číselná hodnota vyjadruje počet riadkov, ktorých je daná bunka súčasťou. Hodnotu `rowspan` sme teda nastavili na 2 a do tejto zlúčenej bunky sme vložili tag pre tlačidlo, ktorému sme nastavili špecifické id. Pomocou tohto id upravujeme formátovanie textu a veľkosti tlačidla v dokumente .css, avšak id je tiež veľmi dôležité pre náš skript, ktorý zabezpečí, aby bolo na tlačidlo možné kliknúť len raz, a to tým spôsobom, že po vykonaní funkcie na výpis inputov nastaví hodnotu jeho CSS parametra `visibility` na `hidden` (prvok na stránke zostáva, avšak už nie je pre užívateľa viditeľný a nie je naň možné kliknúť).

Druhý a tretí blok sú v pôvodnom .html dokumente prázdne, s výnimkou oddeľovacích čiar, avšak majú pridelené id, pomocou ktorých budeme do nich vpisovať.

4.3.2 Získanie počtu premenných a ohraničení

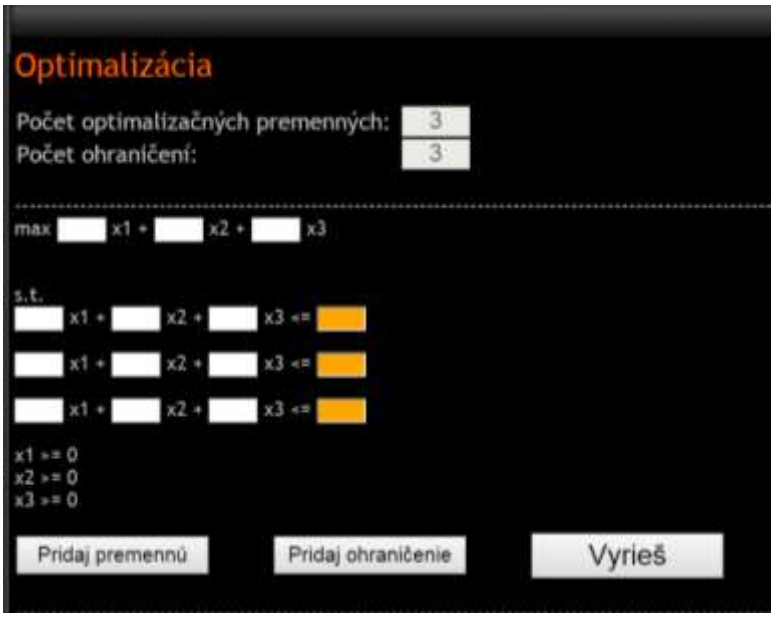
Zistenie počtu premenných a počtu ohraničení je dôležité vo viacerých úrovniach programu. Vytvoríme teda funkciu, ktorú bude možné zavolať v ďalších funkciách a jej výstupom bude číslo – počet premenných. Keďže tento počet je zadaný užívateľom ako číslo do inputu, pre ktorý poznáme id, najjednoduchší spôsob ako zistiť počet premenných je nájsť prvok s týmto id a zistiť jeho hodnotu.

```
function pocet_premennych() {
  var n = document.getElementById("pocet_opt_premennych");
  n.disabled = "disabled";
  return(n.value);
}
```

Obr. 13 - úryvok zdrojového kódu - funkcia na zistenie počtu premenných

To, že výstupom funkcie je daný počet je výhodné preto, aby sme ho v ďalších funkciách mohli uložiť do lokálnej premennej a ďalej s ním pracovať. Nastavenie parametra `disabled` spôsobí, že po prvom spustení tejto funkcie už do inputu nebude možné písať. Toto nastavenie nám ošetruje prípadné problémy, ktoré by v programe mohli nastať, ak by užívateľ spustil niektorú funkciu, a pred spustením ďalšej by zmenil hodnotu počtu v inpute. Počet premenných by teda nebol konzistentný pre všetky funkcie a program by nefungoval správne. Funkcia na zistenie počtu ohraničení pracuje analogicky.

4.3.3 Výpis inputov na plochu



Obr. 14 - inputy na zadávanie argumentov optimalizačného problému

Kliknutím na tlačidlo Input sa pomocou parametra `onClick` zavolá funkcia na výpis inputov. V nej ako prvé uložíme potrebný počet premenných a ohraničení do lokálnych premenných zavolaním príslušných funkcií. Nájdeme na stránke blok medzi oddeľovacími čiarami, do ktorého budeme vpisovať.

```
var blok = document.getElementById("inputy");
var riadok = document.createElement("p");

var id = 'ucelovka_inputs';  // id riadka potrebne na
riadok.id = id;              // pridanie dalsich premennych

blok.appendChild(riadok);
var text = document.createTextNode("max  ")
riadok.appendChild(text);
```

Obr. 15 - úryvok zdrojového kódu – dynamické vpisovanie na stránku

Vpisovanie do bloku uskutočníme tak, že vytvoríme na stránke nový blok (element <p>), ktorý označíme premennou `riadok`. Funkcia `appendChild` nájde posledný HTML element v danom bloku a pripojí zaň argument v zátvorke, pričom týmto argumentom môže byť ľubovoľný element. Pre výpis textu používame funkciu `createTextNode`, ktorá v podstate vytvorí nový element

obsahujúci text, ktorý je argumentom funkcie. `TextNode` však nie je ohraničený tagmi a nemôže vlastniť parametre ako sú *id* alebo *class*.

V ďalšom kroku funkcie sa uskutoční cyklus `for`, ktorý prebehne toľko krát, koľko je premenných. V každom kroku cyklu sa k premennej `riadok` pripíše medzera a za ňu sa pripojí nový input so špecifickým id, ktoré obsahuje číslo kroku cyklu. Za input sa pripíše meno premennej, napr. x1, x2, a pokiaľ sa nejedná o posledný krok cyklu, pripíše sa za meno premennej znamienko plus.

Na rovnakom princípe funguje výpis ohraničení, v tomto prípade využívame dva vnorené `for` cykly – jeden pre počet riadkov, t.j. ohraničení, a vnorený cyklus pre počet premenných. Opäť bude mať každý input pridelené špecifické id, ktoré obsahuje číslo riadka a číslo premennej, ku ktorej patrí. Po pridaní inputov na argumenty premenných – po ukončení vnoreného cyklu – sa pripíše znamienko nerovnosti a input pre pravú stranu. Tomuto takisto priradíme id, ktoré obsahuje číslo riadka, a navyše upravíme jeho .css parametre pre lepší vizuálny efekt výstupu. Na záver pridáme prázdny blok, ktorý označíme id `extra_ohranicenia`, a bude slúžiť na zvyšovanie počtu ohraničení.

4.3.3.1 Tvar zadávania problému

Riešenie problému lineárneho programovania Simplex metódou si vyžaduje vstup v štandardnom tvare. Na (Obr. 14) však vidíme, že požadovaný tvar pre zadanie ohraničení je v tvare nerovností. Dôvod, prečo je tomu tak, vyplýva z cieľov práce, t.j. zjednodušiť zadávanie optimalizačného problému do softwaru, ktorý nám vráti jeho riešenie. Transformácia ohraničení na rovnosti teda funguje automaticky.

Nutnou podmienkou je nezápornosť všetkých optimalizačných premenných, preto automaticky predpokladáme, že problém bude zadaný už v tvare, ktorý túto podmienku spĺňa. V prípade, že potrebujeme zadať problém, ktorý obsahuje neohraničené premenné, musíme sami vykonať substitúciu premenných opísanú v kapitole (1.3.1).

4.3.4 Zvyšovanie počtu premenných a ohraničení

Pri práci s programom by mohla nastať situácia, kedy by bolo potrebné zvýšiť počet premenných alebo ohraničení, v dôsledku nesprávneho riešenia (problém bol zle zadefinovaný a potrebujeme ho upraviť). Keďže počet premenných a ohraničení už je fixne definovaný v inputoch na vrchu stránky, týmto spôsob už výpis inputov pre argumenty problému meniť nemôžeme.

Samozrejme, možným riešením je obnoviť stránku a celý problém zadať znovu. Aby sme však uľahčili užívateľom prácu, pridali sme k programu funkcie, ktoré dávajú možnosť zvýšiť počet premenných alebo ohraničení, a vykreslia na stránku potrebné inputy.

Prvým krokom týchto funkcií je zvýšenie čísla v inpute, v ktorom je uložená hodnota počtu premenných, resp. ohraničení. Táto hodnota je potrebná ako vstup do funkcie na vyriešenie problému.

Princípom dopisovania inputov je nájdenie bloku s požadovaným id (na (Obr. 15) si môžeme všimnúť, že bloku pre argumenty účelovej funkcie sme pridelili id `ucelovka_inputs`). Do tohto bloku pripíšeme input so správnym id, aby sedel počet a poradie premenných. Nové ohraničenia pripisujeme do už spomínaného bloku označeného id `extra_ohranicenia` . Pre pridanie argumentov nového ohraničenia používame cyklus `for`, na koniec pridáme znamienko a input pre pravú stranu.

4.3.5 Uloženie zadaných hodnôt

V tomto momente máme v programe vytvorený priestor, do ktorého užívateľ zadáva optimalizačný problém v štandardnom tvare. Údaje, ktoré do jednotlivých inputov zadá potrebujeme uložiť do globálnej premennej – dvojrozmerného poľa `matica`.

Predtým než začneme niečo ukladať, je potrebné toto pole inicializovať. Na to nám slúži funkcia, ktorá zadefinuje hodnoty jediných troch globálnych premenných v programe.

```
function initialize() {
  S = Number(pocet_ohraniceni()) + Number(pocet_premennych()) + 1;
  R = Number(pocet_ohraniceni()) + 1;
  matica = new Array(R);

  var i = 0;
  for (i; i<R; i++) {
      matica[i] = new Array(S);
  }
}
```

Obr. 16 - úryvok zdrojového kódu - funkcia na inicializáciu veľkosti globálneho poľa

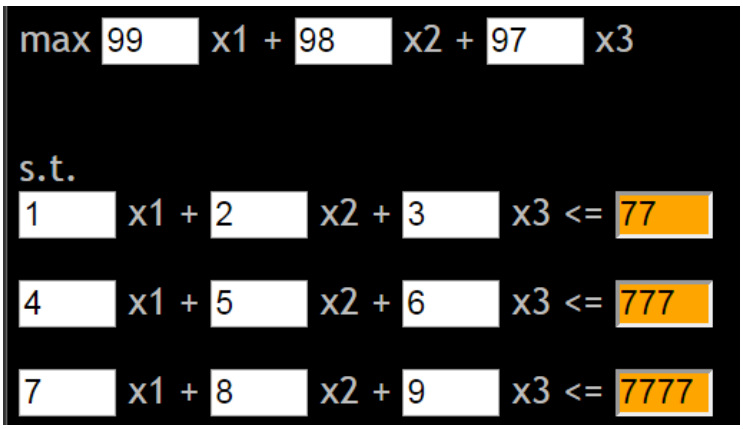
Premenná `R` označuje počet riadkov tabuľky, `S` počet stĺpcov. V jazyku JavaScript využívame na inicializáciu poľa funkciu `new`, ktorá vytvorí nový objekt uvedeného typu. V našom prípade sa do premennej `matica` uloží prázdny stĺpcový vektor (pole), ktorého počet riadkov je `R`. Pomocou cyklu priradíme do každého prvku tohto vektora nové pole – riadkový vektor dĺžky `S` – a vytvoríme tak maticu požadovanej veľkosti (`R x S`).

Matica je teda inicializovaná a potrebujeme ju naplniť hodnotami zadanými v inputoch. Prvým krokom je vyplnenie posledného riadka matice, ktorý zodpovedá účelovej funkcii. Riadok začneme napĺňať potrebnými hodnotami pomocou cyklu, pričom tieto hodnoty získavame pomocou predtým zadefinovaného id parametra. Cyklus prebehne toľko krát, koľko je premenných, zvyšok riadka bude naplnený nulou.

Pokračujeme naplnením posledného stĺpca hodnotami zadanými do inputov pravých strán a ďalej napĺňame maticu argumentmi ohraničení. Toto funguje na podobnom princípe ako výpis inputov, kedy sme každému inputu priraďovali id. Teraz naopak hľadáme prvok s týmto id a ukladáme jeho hodnotu do príslušného riadka a stĺpca poľa `matica`. Keďže medzi konečnými hodnotami v tabuľke a hodnotami na stránke je dobrá symetria (poradie ohraničenia sa zhoduje s poradím riadka tabuľky a číslo premennej s číslom stĺpca), táto úloha sa výrazne zjednodušuje.

Na záver treba do tabuľky doplniť slacky, t.j. doplnkové optimalizačné premenné transformujúce ohraničenia na rovnosti. V podstate do zvyšku tabuľky dopĺňame jednotkovú maticu (n x n) kde n predstavuje počet ohraničení. Dosiahneme to pomocou vnorených cyklov, pričom stĺpcový index ľavého horného rohu matice je počet premenných.

Príklad: Majme takto zadaný optimalizačný problém



Obr. 17 - príklad vstupu zadaného užívateľom

Dvojrozmerné pole `matica (4 x 7)` bude naplnené nasledovne:

Index	0	1	2	3	4	5	6
0	1	2	3	1	0	0	77
1	4	5	6	0	1	0	777
2	7	8	9	0	0	1	7777
3	- 99	- 98	- 97	0	0	0	0

Obr. 18 - ilustrácia spôsobu naplnenia poľa hodnotami zo stránky

- do **zelenej sekcie** sa vpisujú ohraničenia pomocou vnorených cyklov
- do **modrej sekcie** sa vpíšu záporné argumenty účelovej funkcie a zvyšok sa doplní nulami

- **oranžová sekcia** sa naplní argumentmi pravých strán
- zvyšok tabuľky sa naplní jednotkovou maticou

4.3.6 Postup riešenia Simplex tabuľky

Postup riešenia Simplex tabuľky možno rozdeliť do viacerých samostatných krokov:

- nájdenie stĺpca pre pivotovanie
- nájdenie riadka pre pivotovanie
- vydelenie pivotovacieho riadka
- elementárne riadkové operácie
- kontrola optimálnosti

Nakoľko je celá tabuľka uložená v jednej globálnej premennej – `matica`, vytvoríme pre každý uvedený krok riešenia osobitnú funkciu, ktorá bude s touto globálnou premennou pracovať.

a) Nájdenie stĺpca pre iteráciu

Cieľom je nájsť premennú v simplex tabuľke, ktorá najviac ovplyvňuje účelovú funkciu. Keďže argumenty účelovej funkcie predstavujú čísla v poslednom riadku, hľadáme index stĺpca, ktorý má v poslednom riadku najmenšie (najzápornejšie) číslo. Tento stĺpec budeme označovať ako **pivotovací** stĺpec.

b) Nájdenie riadka pre iteráciu

Našli sme premennú, ktorej zvyšovaním sa najrýchlejšie posúvame k optimálnemu riešeniu. Musíme však brať ohľad na ohraničenia a tak hľadáme index riadka spomedzi všetkých riadkov okrem posledného, ktorý má najmenší kladný pomer pravej strany a čísla v stĺpci tejto premennej. Tento riadok budeme nazývať **pivotovací** riadok. Ohraničenie, ktoré pivotovací riadok predstavuje sa stane aktívnym.

c) Vydelenie pivotovacieho riadka

Prvok matice, ktorý je priesečníkom pivotovacieho riadka a stĺpca, sa musí rovnať 1. To dosiahneme tak, že vydelíme každý prvok riadka daným číslom (jedná sa o korektnú matematickú operáciu, keďže riadok predstavuje rovnicu – súčet premenných sa rovná pravej strane).

d) Elementárne riadkové operácie

Cieľom elementárnych riadkových operácii je vytvoriť z pivotovacieho stĺpca stĺpec bázickej premennej. V pivotovacom riadku teda zostáva číslo 1, ktoré sme získali v predošlom kroku. V ostatných riadkoch musí v tomto stĺpci zostať 0.

```
for (i=0; i<R; i++) {
   if (i!=inx1) {                //inx1 je riadkový index
      X = matica[i][inx2];       //inx2 je stĺpcový index
      for (j=0; j<S; j++) {      // prvku na pivotovanie
        matica[i][j] += (-X * matica[inx1][j]);
      }
   }
}
```

Obr. 19 - úryvok zdrojového kódu - pivotovanie

Pomocou vnorených `for` cyklov prechádzame cez všetky riadky okrem pivotovacieho, a v každom riadku cez všetky prvky. Pre každý riadok uložíme hodnotu čísla v pivotovacom stĺpci do premennej `X`. Od všetkých stĺpcov riadka potom odrátame `X` násobok čísla v pivotovacom riadku v danom stĺpci. Takto dosiahneme, aby sa v pivotovacom stĺpci všetky prvky rovnali 0 (odčítame od nich 1-násobok seba samých).

e) Kontrola optimálnosti

Predošlým krokom sa zmenili aj jednotlivé prvky v poslednom riadku. Ak sa v poslednom riadku nachádza nejaké záporné číslo, znamená to, že zvyšovaním hodnoty príslušnej premennej pre daný stĺpec sa zvýši hodnota účelovej funkcie. Z toho jasne vyplýva, že takéto riešenie nemôže byť optimálne. V takomto prípade musí program znovu pristúpiť k prvému kroku.

Týchto päť funkcií pracuje vo vnútri cyklu `while`, ktorý začína vykreslením tabuľky. Posledný krok cyklu rozhoduje o tom, či sa má cyklus opakovať alebo ukončiť. Ak je nájdené riešenie optimálne a cyklus sa ukončí, treba zistiť optimálne hodnoty premenných.

4.3.7 Zistenie hodnôt v optime

V poslednom riadku tabuľky sa už nenachádza záporné číslo a hodnota v pravom dolnom rohu predstavuje optimálnu hodnotu účelovej funkcie.

Zadefinujeme si jednorozmerné pole `OPT`, ktoré bude predstavovať optimálne hodnoty premenných (počet prvkov poľa sa teda musí zhodovať s počtom premenných). To znamená, že každý stĺpec tabuľky, okrem stĺpca pravej strany, má priradený jeden prvok z poľa `OPT`, pričom sa ich stĺpcové indexy číselne rovnajú (pre premennú v stĺpci `[0]` je optimálna hodnota uložená v prvku `OPT[0]`).

Aby sme zistili hodnoty premenných v optime, musíme najprv určiť, ktoré premenné sú bázické. Túto úlohu vyriešime pomocou vnorených `for` cyklov.

Pre každý stĺpec vopred predpokladáme, že ide o stĺpec bázickej premennej. V tele druhého `for` cyklu sa nachádza podmienka, ktorá kontroluje, či sa jedná o bázickú premennú.

Ak áno, nasleduje ďalšia podmienka, ktorá ak sa prvok rovná 1 (a je to prvý výskyt čísla 1 v danom stĺpci), uloží do príslušného prvku poľa `OPT` hodnotu pravej strany v aktuálnom riadku. Ak podmienka nie je splnená, kontrolovaný prvok sa musí rovnať nule. V prípade, že tomu tak nie je, označíme premennú ako nebázickú a do príslušného prvku `OPT` sa uloží 0.

Po ukončení vnorených cyklov vypíšeme optimálne hodnoty, spolu s hodnotou účelovej funkcie na plochu pre užívateľa, pričom na vyriešenie tejto úlohy nám vystačí jednoduchý cyklus `for`.

4.3.8 Vykreslenie Simplex tabuľky

Po každej iterácii smerom k optimálnemu riešeniu, kde sa menili hodnoty v simplex tabuľke, je potrebné túto tabuľku vykresliť. Cieľom riešenia totiž nie je samotný výsledok, ale ilustrácia postupu, ako sa k výsledku dopracovať.

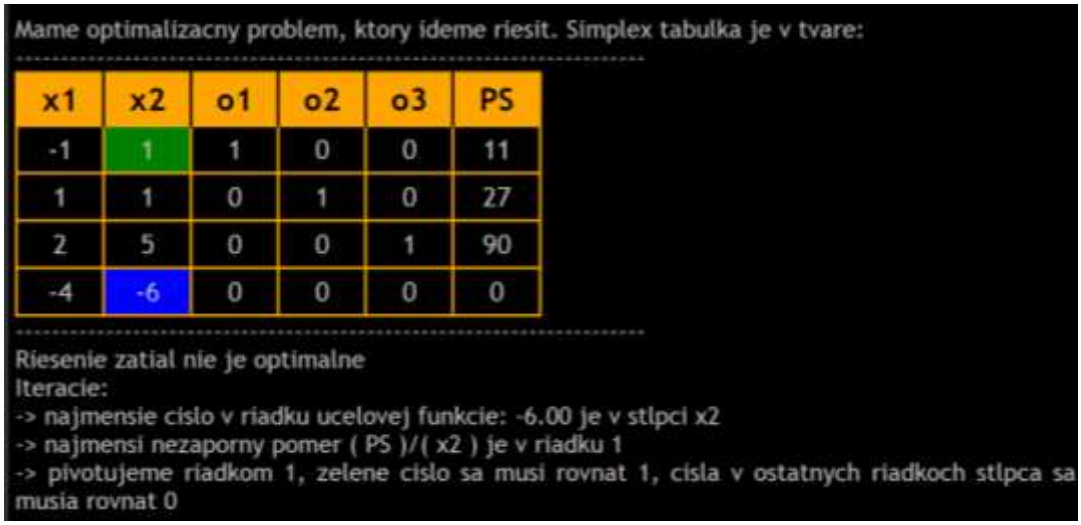
Funkcia pre vykreslenie tabuľky pracuje na báze cyklov a je volaná v rámci `while` cyklu, ktorý tvorí jadro funkcie volanej tlačidlom *Vyrieš*. Po zavolaní funkcie postupuje program nasledovným spôsobom:

- Vytvoríme nový blok `<p>` a doň vložíme novú tabuľku – element `<table>`
- riadky pridávame využitím funkcie `insertRow(n)`, do riadka pridávame potrebný počet buniek využitím cyklu a funkcie `insertCell(m)`. Parametre funkcií `n` a `m` označujú miesto (index) kam sa má daný element pridať. Využívame preto argumenty cyklov, ktoré štandardne označujeme písmenami `i` a `j`.
- do každej bunky sa vpíše zaokrúhlená číselná hodnota príslušného prvku poľa `matica` – syntax takéhoto príkazu má tvar:

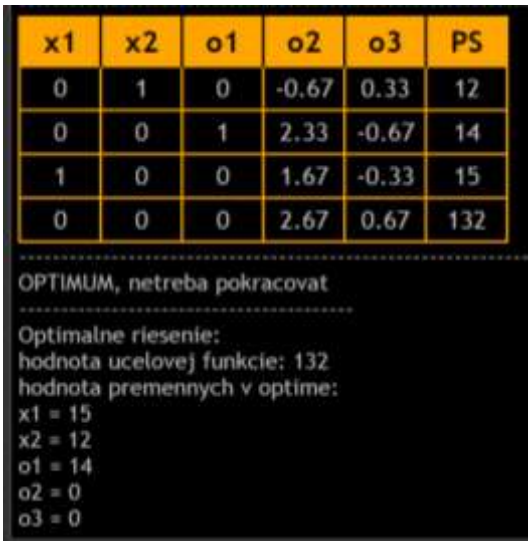
```
bunka.innerHTML = Number((matica[i][j]).toFixed(2));
```

- Pre každý riadok uložíme všetky bunky do premennej `bunky` využitím príkazu `getElementsByTagName("td")`. Premenná `bunky` teraz predstavuje pole, pričom k jednotlivým prvkom tohto poľa možno pristupovať pomocou indexov (prvky sú zoradené v tom istom poradí, v akom tvoria HTML štruktúru stránky). Využitím cyklu potom upravíme .css parametre každej bunky – hrúbka a farba ohraničenia, veľkosť a zarovnanie textu. Tu sa nachádza aj podmienka, pomocou ktorej farebne rozlíšime pivotovací riadok a stĺpec tabuľky od ostatných buniek na základe indexov.
- Na záver, keď sú naplnené všetky riadky, pridáme nový riadok na začiatok tabuľky – hlavičku. Postupne ju naplníme bunkami využitím dvoch cyklov, ktorými rozdelíme stĺpce

tabuľky podľa toho, či sa jedná o optimalizačnú premennú (x), alebo slack (o). Posledná bunka označuje stĺpec pravej strany. Nastavíme formátovanie pre hlavičku, rovnakým spôsobom ako formátovanie jednotlivých riadkov tabuľky.



Obr. 20 - ilustrácia vykreslenia tabuľky a začiatok riešenia zadaného problému



Obr. 21 - posledná tabuľka a optimálne riešenie

5 Záver

Cieľom práce bolo vytvoriť software, ktorý by riešil problémy lineárnej optimalizácie, a možno povedať, že tento cieľ bol splnený. Užívateľ najskôr určí veľkosť problému, ktorý potrebuje vyriešiť. Následne sa mu na ploche stránky zobrazí pole, do ktorého jednoducho a prehľadne zadá argumenty celého problému. V prípade, že veľkosť problému určil nesprávne, má možnosť ho rozšíriť o nové ohraničenie, prípadne novú premennú.

Software teda funguje pre ľubovoľne veľké optimalizačné problémy a zobrazuje postup riešenia Simplexovou metódou – výstupom nie sú iba hodnoty optimálneho riešenia, ale zobrazí sa tabuľka pre každú iteráciu cez zlučiteľné bázické riešenia aj s informáciou ako pri riešení ďalej postupovať.

Čo sa týka využiteľnosti programu, má potenciál stať sa dobrou pomôckou pri výučbe predmetu Optimalizácia, predovšetkým v časti venovanej lineárnemu programovaniu, a pomôcť študentom lepšie pochopiť preberané učivo a simplexovú metódu. Môžu si tu totiž overiť správnosť ručnej implementácie simplexovej metódy, rovnako ako správnosť ich formulácie daného optimalizačného problému. Navyše program umožňuje riešiť problémy lineárneho programovania bez potreby MATLAB-u.

Existuje však ešte mnoho vecí, ktoré je na programe možné vylepšiť. Patrí medzi ne napríklad možnosť zadávať problémy s neohraničenými optimalizačnými premennými, možnosť voľby znamienka pre každé ohraničenie osobitne, prípadne možnosť znížiť počet ohraničení alebo premenných problému. Na jeho vývoji sa preto plánuje ďalej pracovať, aby práca s ním bola ešte jednoduchšia a prehľadnejšia.

Software bol síce vytvorený hlavne pre účel výučby a vizualizácie postupu riešenia, no funkcie v zdrojovom kóde, pomocou ktorých sa program dopracuje k optimálnemu riešeniu, by bolo možné exportovať aj do iného softwaru. Vyžadujú totiž iba tri vstupy: počet premenných, počet ohraničení, a maticu v tvare simplex tabuľky, čo nie je ťažké získať pre akýkoľvek lineárny problém. Iný software by potom mohol slúžiť napríklad na optimalizované riadenie nejakého procesu, ktoré by bolo možné upravovať a riadiť cez akékoľvek mobilné zariadenie (smartphone alebo tablet), na ktorom je problém nainštalovať MATLAB, no JavaScript funguje automaticky.

Zoznam použitej literatúry

BEREŽNÝ Š., KRAVECOVÁ D. – Lineárne programovanie, Košice: Technická univerzita v Košiciach, 2012 – ISBN 978-80-553-0910-1

MURTY, K. – Linear programming. New York: John Wiley & Sons Inc. 1983 – ISBN 978-04-710-9725-9

BOYD, S. – Convex Optimization, Cambridge: Cambridge University Press 2004 – ISBN 978-0-521-83378-3

Prílohy

Príloha: CD médium – práca v elektronickej podobe, prílohy v elektronickej podobe.