

# **Web-aplikace pro řešení problému dynamického programování**

Web-application Solving the Dynamic Programming Problem

Bc. Pavel Machourek

---

Diplomová práce  
2013



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Pavel Machourek**  
Osobní číslo: **A11842**  
Studijní program: **N3902 Inženýrská informatika**  
Studijní obor: **Automatické řízení a informatika**  
Forma studia: **kombinovaná**

Téma práce: **Web-aplikace pro řešení problému dynamického programování**

## Zásady pro vypracování:

1. **Důkladně se seznamte se speciální metodou optimalizace Dynamické programování a vypracujte literární rešerši na uvedené téma.**
2. **Navrhněte a realizujte interaktivní webovou aplikaci, která umožní vyřešit tabulkovou formou úlohu dynamického programování pomocí principu dělení zdrojů.**
3. **Po zadání vstupních dat uživatelem aplikace poskytně informace o řešení zadaného problému, které dovolí najít jednorázově nebo s možností zobrazovat jednotlivé kroky řešení.**
4. **Zobrazte průběh i výsledek řešení v přehledné výstupní sestavě a uložte ji také ve formátu PDF.**
5. **Vytvořenou aplikaci důkladně otestujte a na vybraných problémech demonstруйте správnost řešení.**
6. **Věnujte dostatečnou pozornost zabezpečení aplikace.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **KOŘENÁŘ, Václav a Milada LAGOVÁ. Optimalizační metody. Praha: Oeconomica, 2003. ISBN 80-245-0609-2.**
2. **SAMEK, Jaroslav, Zdeňka NEČASOVÁ a Jan KODERA. Dynamické programování v ekonomických procesech. Praha: SPN, 1989. ISBN 80-707-9888-2.**
3. **HÁJEK, Michal. Soubor úloh ke cvičení do předmětu Optimalizace – lineární a dynamické programování. Zlín, 2009. Bakalářská práce. Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky.**
4. **GAJDA, Bohumil. Speciální metody optimalizace. Zlín, 2009. Diplomová práce. Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky.**
5. **ESPOSITO, Dino. ASP.NET a ADO.NET: tvorba dynamických webových stránek. Praha: Grada, 2003. ISBN 80-247-0474-9.**

Vedoucí diplomové práce:

**doc. Ing. František Gazdoš, Ph.D.**

Ústav řízení procesů

Datum zadání diplomové práce:

**24. února 2013**

Termín odevzdání diplomové práce:

**11. června 2013**

Ve Zlíně dne 24. února 2013



prof. Ing. Vladimír Vašek, CSc.

*děkan*



prof. Ing. Vladimír Vašek, CSc.

*ředitel ústavu*

## **ABSTRAKT**

Tato práce se zabývá optimalizační metodou Dynamické programování. Nejprve popisuje a klasifikuje problémy řešené pomocí optimalizačních metod. Dále podrobněji rozebírá metody a postupy dynamického programování a zvláště se zaměřuje na problém dělení zdrojů. Ve druhé části je popsán návrh a implementace webové aplikace, která řeší problém dělení zdrojů tabulkovou metodou.

Klíčová slova: Optimalizace, Dynamické programování, Nelineární programování, Dělení zdrojů, Webová aplikace, ASP.

## **ABSTRACT**

This thesis deals with the optimization method dynamic programming. It describes and classifies problems solved by optimization methods. Methods and approaches of Dynamic programming are described in detail, especially the problem of Resource allocation. Design and implementation of a web application solving the Resource allocation problem in the table form are described in its second part.

Keywords: Optimization, Dynamic programming, Nonlinear programming, Resource allocation, Web application, ASP.

Na tomto místě bych rád poděkoval doc. Ing. Františku Gazdošovi, Ph.D. za cenné připomínky a odborné rady, kterými přispěl k vypracování této diplomové práce.

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....  
podpis diplomanta

**OBSAH**

<b>ÚVOD</b> .....	<b>8</b>
<b>I TEORETICKÁ ČÁST</b> .....	<b>9</b>
<b>1 OPTIMALIZACE</b> .....	<b>10</b>
1.1 ZÁKLADNÍ POJMY .....	10
1.2 OPTIMALIZAČNÍ METODY .....	11
<b>2 DYNAMICKÉ PROGRAMOVÁNÍ</b> .....	<b>14</b>
2.1 FIBONACCIHO POSLOUPNOST.....	15
2.2 DIJKSTRŮV ALGORITMUS .....	16
2.3 HANOJSKÉ VĚŽE .....	19
2.4 DĚLENÍ ZDROJŮ .....	21
2.4.1 Příklad .....	22
2.4.2 Příklad se substitucí.....	23
2.4.3 Existující aplikace řešící problém dělení zdrojů .....	25
<b>3 WEBOVÉ APLIKACE</b> .....	<b>27</b>
3.1 APACHE.....	28
3.2 INTERNET INFORMATION SERVICES.....	29
3.3 JAVA .....	29
3.4 BEZPEČNOST .....	30
3.4.1 Injection.....	30
3.4.2 XSS - Cross-site scripting .....	31
<b>II PRAKTICKÁ ČÁST</b> .....	<b>32</b>
<b>4 NÁVRH ŘEŠENÍ PROBLÉMU DĚLENÍ ZDROJŮ</b> .....	<b>33</b>
4.1 PRINCIP ALGORITMU .....	33
4.2 VOLBA PLATFORMY .....	35
<b>5 POPIS IMPLEMENTACE WEBOVÉ APLIKACE</b> .....	<b>36</b>
5.1 INTERNÍ STRUKTURA APLIKACE.....	36
5.1.1 Získání údajů .....	37
5.1.2 Kontrola vstupu .....	37
5.1.3 Výpočet řešení.....	38
5.1.4 Výpis řešení.....	39
5.2 UŽIVATELSKÉ ROZHRAŇÍ .....	40
5.3 TESTOVÁNÍ.....	44
<b>6 ŘEŠENÉ PŘÍKLADY</b> .....	<b>45</b>
<b>ZÁVĚR</b> .....	<b>51</b>
<b>ZÁVĚR V ANGLIČTINĚ</b> .....	<b>52</b>
<b>SEZNAM POUŽITÉ LITERATURY</b> .....	<b>53</b>
<b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK</b> .....	<b>55</b>
<b>SEZNAM OBRÁZKŮ</b> .....	<b>56</b>
<b>SEZNAM TABULEK</b> .....	<b>57</b>
<b>SEZNAM PŘÍLOH</b> .....	<b>58</b>

## ÚVOD

Optimalizace se zabývá hledáním vstupních hodnot, pro které má požadovaná funkce minimální nebo maximální hodnotu. Aniž bychom to věděli, řešíme optimalizační úlohy každý den. Která cesta je nejkratší? Vyplatí se mi na dojíždění do práce vlastní automobil? Je pro mě vhodnější levnější inkoustová tiskárna, nebo dražší laserová? Do čeho investovat, abych nejvíce vydělal? Abychom si na tyto otázky mohli odpovědět, musíme podvědomě vyřešit optimalizační úlohu. Na některé otázky jsme schopni si odpovědět na základě vlastní zkušenosti nebo úsudku, na ty složitější ale potřebujeme matematický aparát pro popis a vyřešení daného problému.

Optimalizační úlohy jsou různého typu, stejně tak metody a postupy, které je řeší. Jednou z nich je dynamické programování, které vychází z principu optimality. Za jejím vznikem stojí Richard Bellman. Tato metoda spočívá v rozdělení komplexního problému na jednodušší podproblémy, ty pak řešit a z nich sestavit celkové optimální řešení. Mimo jiné se tímto postupem hledá minimum nebo maximum nelineárních funkcí. Při tom se postupně optimalizují jednotlivé separované funkce a tím se problém zjednoduší na jednorozměrnou optimalizaci. Praktické využití těchto postupů se používá například v ekonomii a fyzice.

V teoretické části této práce se blíže seznámíme s historií a principy dynamického programování. Také si názorně ukážeme několik algoritmů, které staví na principech dynamického programování, a nahlédneme do problematiky webových aplikací. To proto, aby mohla být v praktické části navržena a implementována webová aplikace řešící problém dělení zdrojů tabulkovou formou. Tato aplikace je umístěna na přiloženém DVD.

## **I. TEORETICKÁ ČÁST**

## 1 OPTIMALIZACE

Matematická optimalizace (nebo také matematické programování) se zaměřuje na vyhledávání extrémů funkcí nebo také na nalezení nejlepšího prvku z množiny podle zadaných kritérií. S těmito úlohami se velmi často setkáváme i v praktickém životě, jen jsou formulovány slovně a řešíme je na základě zkušeností nebo intuice. Tento způsob řešení ale není použitelný pro složité situace, a je tedy potřeba mít k dispozici matematický aparát, schopný tyto úlohy popsat a vyřešit.

V nejjednodušším možném případě se jedná o nalezení minima nebo maxima funkce reálné proměnné postupným dosazováním hodnot z definičního oboru funkce a výpočtem hodnoty funkce. Zobecněním teorií a postupů zasahuje optimalizace velkou část aplikované matematiky (oblasti jako statistika, výpočetní technika, ekonomika, doprava, teorie grafů, teorie her).

Dále v textu budou často používány některé pojmy specifické pro oblast optimalizace. Pro předejití případných nedorozumění si je nadefinujeme hned na začátku.

### 1.1 Základní pojmy

**Funkce  $f$**  je jednoznačné přiřazení prvků z definičního oboru funkce  $D_f$  k prvkům z množiny oboru hodnot  $R_f$ .

$$f: D_f \rightarrow R_f \quad (1)$$

Většinou funkci budeme zapisovat ve tvaru  $y=f(x)$ , kde  $x$  je argument funkce a prvkem množiny  $D_f$ ,  $y$  je prvkem  $R_f$ . Další možný zápis funkce je  $f(x_1, \dots, x_n)$  značící funkci více proměnných.

**Účelová funkce  $f(x_1, \dots, x_n)$**  je zadaná funkce reálných proměnných, jejíž optimalizací se zabýváme – tj. hledáme její extrémy.

**Extrémy funkce** lze rozdělit na lokální a globální. Lokální extrémy jsou definovány na omezeném intervalu, globální extrémy platí pro celý definiční obor funkce. Extrémy mohou být dva - minimum a maximum.

Hledáme-li extrémy  $n$ -rozměrné funkce na celém prostoru  $\mathbb{R}^n$ , jedná se o takzvaný volný extrém. Pokud nehledáme extrémy v celém prostoru  $\mathbb{R}^n$ , ale tento prostor zmenšíme pomocí **omezujících podmínek**, mluvíme o vázaném extrému. Pokud jsou omezující podmínky ve tvaru

$$g_j(x_1, \dots, x_n) = 0; j = 1, \dots, m; m < n \quad (2)$$

jedná se o omezující podmínky typu rovnost. U podmínek ve tvaru

$$g_j(x_1, \dots, x_n) \geq 0; j = 1, \dots, m; x_i \geq 0; i = 1, \dots, n \quad (3)$$

se bavíme o omezujících podmínkách typu nerovnost. [1]

Pomocí těchto pojmů můžeme rozdělit úlohy optimalizace do následujících skupin: [2]

Volný extrém

- Jednorozměrný - Reálná funkce jedné reálné proměnné
- Vícerozměrný - Reálná funkce více reálných proměnných

Klasický vázaný extrém

- Reálná funkce více reálných proměnných, omezující podmínky typu rovnost

Neklasický vázaný extrém

- Reálná funkce více reálných proměnných, omezující podmínky typu nerovnost

Teorie her a optimální rozhodování

- Volba vhodné strategie pro docílení nejlepšího výsledku

## 1.2 Optimalizační metody

Optimalizační metody lze rozdělit do několika skupin [1]

**Analytické metody**

- Jednorozměrné případy
- Vícerozměrné případy
- Omezení typu rovnost
- Omezení typu nerovnost

Analytické metody při zkoumání extrémů vycházejí z matematických definic. Pracují s derivacemi funkce a jejich vlastnostmi. První derivace určí body podezřelé z extrému, druhá derivace může odhalit typ extrému. U vícerozměrných metod se pak pracuje s gradienty (vektor parciálních derivací) a Hessovou maticí (druhé parciální derivace).

### Iterační metody

- Komparativní
- Gradientní
- Metody náhodného vyhledávání

Iterační metody jsou založeny na principu hledání extrému postupným výpočtem hodnot a přibližováním se k hledanému extrému. První iterace často získají jen velmi hrubý přehled o průběhu zkoumané funkce, ale odhadnou, ve kterých místech by se extrémy mohly nacházet. Další iterace již hledají extrém v místech, kde je očekáván a zpřesňují jeho odhad. To, kolik iterací je potřeba pro dosažení požadované přesnosti, určuje rychlost konvergence zvolené metody. O ukončení výpočtu je pak nejčastěji rozhodováno na základě požadované chyby nebo přesnosti výpočtu.

### Speciální metody

- Lineární programování
- Nelineární programování
  - Kvadratické programování
  - Separovatelné programování
    - Dynamické programování
  - Konvexní programování

Speciální optimalizační metody se zabývají jen specifickým typem zkoumané účelové funkce a tvarem omezujících podmínek. Lineární programování umožňuje najít extrém lineární účelové funkce s větším počtem lineárních omezujících podmínek, kvadratické programování pracuje s lineárními podmínkami a kvadratickou účelovou funkcí, konvexní programování má omezení i účelovou funkcí konvexní a dynamické programování pracuje s nelineární účelovou funkcí a lineární omezující podmínkou.

### Teorie her

- Maticové hry
- Diferenciální hry

Teorie her se zabývá volbou optimálního rozhodnutí v případě konfliktu nebo spolupráce inteligentně se rozhodujících hráčů. Používají se pak různé klasifikace her, například podle úplnosti nebo neúplnosti znalostí o hře, zda hráči spolupracují nebo soupeří, jestli hra používá diskrétní nebo spojitý čas. Zápis her bývá realizován normální (tabulkovou) nebo extenzivní (stromovou) formou. [3]

## 2 DYNAMICKÉ PROGRAMOVÁNÍ

Pojem dynamické programování vymyslel a zavedl do praxe Richard Ernest Bellman ve čtyřicátých letech dvacátého století, když se zabýval řešením problémů, kdy je potřeba hledat jedno nejlepší řešení za druhým. Poté, co se v letech 1944-1946 účastnil projektu Manhattan a v roce 1946 získal titul Ph.D. na Princetonské univerzitě, pracoval jako docent matematiky na univerzitě ve Stanfordu. V létě roku 1949 také začal pracovat v neziskové organizaci RAND (Research and Development) na projektu zabývajícím se procesy s víceúrovňovým rozhodováním. Tuto organizaci financovalo letectvo Spojených Států amerických a to spadalo pod ministerstvo obrany. Pro svůj matematický výzkum potřeboval vymyslet název, který by "maskoval jeho činnost" před tehdejším ministrem obrany Wilsonem, který nesnášel slovo výzkum. S pojmem matematický na tom nebyl lépe.

Slovo programování Bellman použil ve významu plánování a rozhodování (nikoliv v dnešním informatickém významu návrhu a implementace algoritmu). Poté se snažil přijít s přídatným jménem, které by popisovalo víceúrovňové, časově proměnné děje, ale nepodařilo se mu najít vhodnější pojem než dynamický. Přestože chtěl přijít s výstižnějším označením, nic vhodnějšího ho nenapadlo a byl spokojen i se souslovím dynamické programování. V roce 1953 význam tohoto pojmu pozměnil do dnešního významu - řešení složitých problémů pomocí jejich rozkladu na jednodušší vnořené problémy. Vychází při tom z tzv. principu optimality [4]:

"Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must contribute an optimal policy with regard to the state resulting from the first decisions." [5]

Tento princip by se dal volně přeložit jako "Optimální strategie se skládá z optimálních podstrategií.". Pokud tedy problém rozdělíme na dílčí části, a tyto jednotlivé části vyřešíme optimálně, celkové řešení bude opět optimální. Toto pravidlo bylo později na počest Richarda Bellmana a jeho zásluhám v této oblasti pojmenováno po něm - Bellmanův princip optimality.

Z obecného pohledu matematiky, výpočetní techniky a ekonomiky je dynamické programování způsob řešení složitých problémů pomocí rozkladu na jednodušší podproblémy. Tyto podproblémy musí vykazovat znaky překrývajících se podproblémů a optimální podstruktury.

Problém má překrývající se podproblémy, pokud lze problém rozložit na podproblémy takovým způsobem, že řešení těchto podproblémů můžeme několikrát využít a nemusíme ho několikrát počítat.

Optimální struktura je, když lze optimální řešení sestavit z nezávislých optimálních řešení identifikovaných podproblémů. Například nejkratší cesta z Hradce Králové do Olomouce vede přes Vysoké Mýto a Mohelnici. Řešení tak lze složit z nezávislých nejkratších tras z Hradce do Mýta, z Mýta do Mohelnice a z Mohelnice do Olomouce. Nejkratší trasa z Olomouce do Mohelnice nezávisí na trase z Hradce do Mýta. Takovou závislost lze nalézt například u letenek. Kupujeme totiž letenku na celý let s mezipřistáními, ne několik nezávislých letenek. Samostatné letenky mohou být nejlevnější z Prahy do Londýna a z Londýna do New Yorku, ale zakoupená letenka z Prahy do New Yorku bude levnější s mezipřistáním v Paříži. [6]

Právě separovatelnost podproblémů a potřeba uchování výsledků předchozích výpočtů odlišuje dynamické programování od běžných rekurzivních a iteračních algoritmů.

## 2.1 Fibonacciho posloupnost

Fibonacciho posloupnost je řada čísel, z nichž se každé číslo získá součtem dvou předcházejících. První dvě hodnoty jsou 0 a 1. Řada tedy začíná čísly 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Pokud budeme chtít fibonacciho posloupnost implementovat čistě podle definice, bude algoritmus vypadat takto [6]:

```
1 int fibonacci (int n) {
2     if n=0 return 0;
3     if n=1 return 1;
4     return fibonacci(n-1)+fibonacci(n-2);
5 }
```

Tento algoritmus má exponenciální asymptotickou časovou složitost  $O(X^N)$ , jeho výpočet je pro vyšší hodnoty  $n$  časově velmi náročný. Můžeme ho ale modifikovat podle zásad,

kteřé vycházejí z dynamického programování - tj pamatovat si mezivýsledky a ty pak používat při dalším postupu. Neřešit celý problém najednou, ale postupně řešit dílčí podproblémy a jejich optimální řešení použít pro sestavení celkového řešení. Tímto způsobem modifikovaný algoritmus bude mít následující podobu:

```
1 int fibonacci (int n) {
2     if n=0 return 0;
3     if n=1 return 1;
4     int new, previous=0, current=1;
5     for i=2 to n {
6         new=previous+current;
7         previous=current;
8         current=new;
9     }
10    return current;
11 }
```

Tento optimalizovaný algoritmus již má lineární časovou složitost a je tak mnohem vhodnější pro výpočet více členů Fibonacciho posloupnosti.

## 2.2 Dijkstrův algoritmus

Principy dynamického programování se také velmi často používají v souvislosti s vyhledáváním nejkratší cesty v grafu. Sám Richard Bellman se touto problematikou zabýval, je autorem algoritmu pro vyhledávání nejkratší cesty v ohodnoceném grafu z jednoho uzlu, i při záporném ohodnocení hran (1958). S tímto algoritmem ve stejné době kromě něho přišel i Lester Ford (1956) a Edward Moore (1957), bývá proto nazýván Bellmanův-Fordův-Moorův algoritmus. S podobným algoritmem, který je sice rychlejší, ale neporadí si se záporně ohodnocenými hranami, přišel také Edsger Dijkstra v roce 1956.

Bellmanův princip optimality přeformulovaný do teorie grafů zní:

$$u(x, z) = \min(u(x, y) + u(y, z)) \quad (4)$$

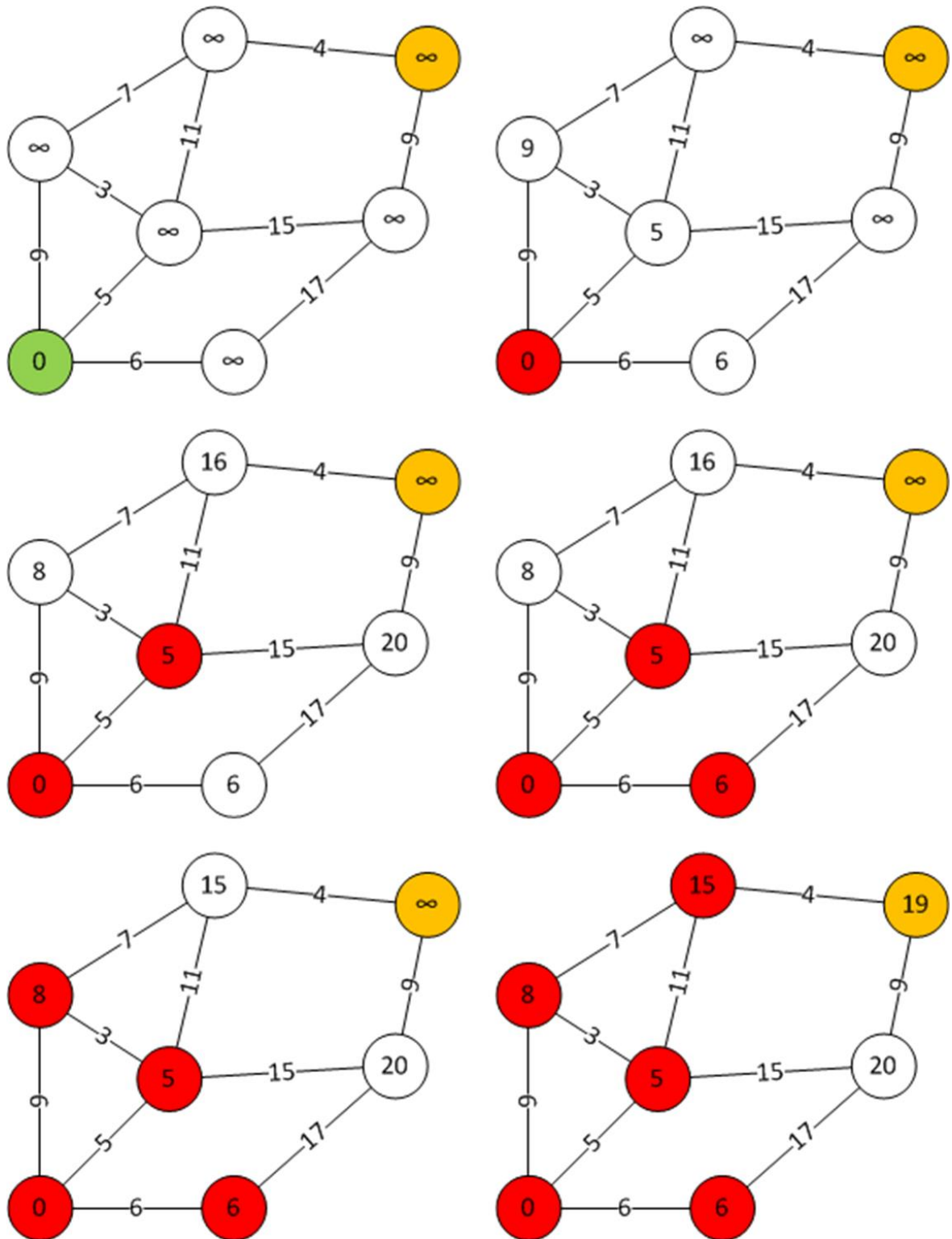
kde funkce  $u$  značí nejkratší cestu mezi dvěma uzly.

Každá nejkratší cesta se tedy skládá z nejkratších cest - nejkratší cesta  $\{x, \dots, y, \dots, z\}$  mezi uzly  $x$  a  $z$  obsahuje také nejkratší cestu z  $x$  do  $y$  a z  $y$  do  $z$ .

Dijkstrův algoritmus pro vyhledání nejkratší trasy v ohodnoceném grafu [7]:

1. Hodnota všech uzlů, kromě výchozího, se nastaví na nekonečno (tj. neznáme cestu)
2. Všechny uzly se označí jako nenavštívené, výchozí uzel se označí jako aktuální
3. Z aktuálního uzlu projdeme všechny sousedící nenavštívené uzly a spočítáme k nim ohodnocení cesty. Pokud je nižší než hodnota uzlu, použije se nová hodnota. Tyto sousedící uzly pořád zůstávají označené jako nenavštívené.
4. Aktuální uzel označme za navštívený. Navštívené uzly už dále nebudeme kontrolovat.
5. Skončíme, pokud všechny nenavštívené uzly mají hodnotu nekonečno, nebo jsme dosáhly cílového uzlu a žádný nenavštívený uzel nemá menší ohodnocení, než hodnota cílového uzlu.
6. Vybereme nenavštívený uzel s nejnižším ohodnocením, označíme ho jako aktuální a přejdeme k bodu 3.

Praktický výpočet nejkratší trasy je znázorněn na obrázku 1



Obr. 1. Ilustrace určení nejkratší trasy pomocí Dijkstrova algoritmu

### 2.3 Hanojské věže

Hanojské věže jsou matematický hlavolam pocházející z 19. století. Skládá se ze tří kolíků (věží), jeden z nich je počáteční, druhý pomocný a třetí cílový. Na počátečním kolíku jsou kotouče různé velikosti seřazené od největšího dole po nejmenší nahoře. Úkolem je přesunout všechny kotouče z počátečního kolíku na cílový. Hlavolam má následující pravidla:

- Najednou lze přemístit pouze jeden kotouč
- Žádný kotouč nesmí být položen na menší kotouč
- Jeden tah tedy znamená vzít vrchní kotouč z některého kolíku a umístit ho na jiný kolík, na kterém jsou pouze větší nebo žádné kotouče.

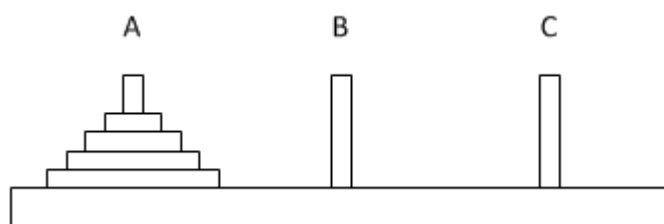
Řešení této hádanky je opět rekurzivní. Pro názornost budeme uvažovat situaci se třemi kotouči. Hlavní problém této hádanky je přesunutí největšího spodního kotouče (který je potřeba uvolnit a nemůže se položit na jiné kotouče) z počáteční tyče na cílovou. Kvůli tomu potřebujeme mít počáteční i cílovou věž volnou, musíme tedy dva menší kotouče přesunout na pomocnou tyč. Pro přesunutí dvou menších kotoučů na pomocnou tyč řešíme opět ten samý problém. Potřebujeme uvolnit prostřední kotouč, aby nebyl blokován nejmenším. Nejmenší kotouč tedy přesuneme na cílovou tyč, tím uvolníme počáteční a pomocnou tyč pro přesun prostředního kotouče a nejmenší kotouč poté přesuneme na tento střední. Tím máme uvolněný největší kotouč pro přesun na cílovou tyč a opět řešíme, jak dostat zbývající dva kotouče z pomocné tyče na cílovou. Obdobný postup bude platit pro libovolný větší počet kotoučů. [8]

Rekurzivně by se dalo řešení zapsat ve tvaru:

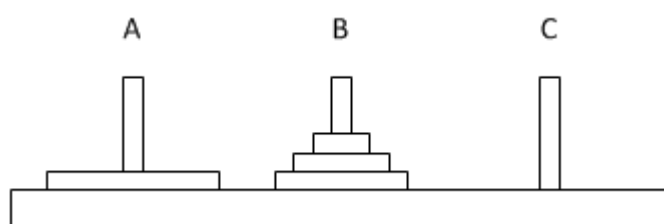
```
1 void Hanoj (int n, char from, to, temp) {  
2     Hanoj(n-1, from, temp, to);  
3     print("Presun z " & from & "na" & to);  
4     Hanoj(n-1, temp, to, from);  
5 }
```

Na první pohled tento algoritmus nevykazuje vlastnosti dynamického programování, ukládání hodnot zde totiž není vyjádřeno explicitně. Je zde totiž vyjádřeno implicitně pomocí druhého rekurzivního volání s prohozenými parametry. Ilustrace jedné úrovně

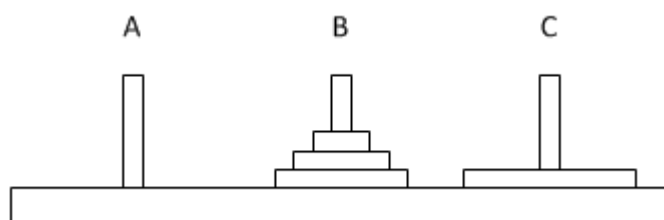
tohoto algoritmu s provedením obou rekurzivních volání jako jednoho kroku je znázorněno na obrázcích 2,3,4 a5.



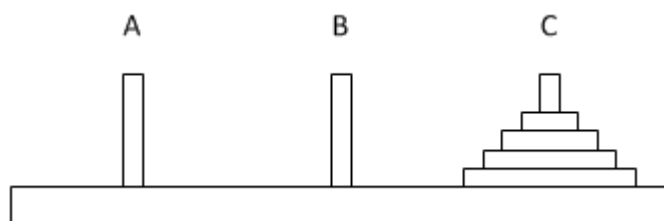
*Obr. 2. Výchozí stav*



*Obr. 3. Uvolnění spodního kotouče přesunutím všech ostatních*



*Obr. 4. Poté přesunout největší kotouč*



*Obr. 5. A nakonec zbývající kotouče přesunout na cílový kolík*

## 2.4 Dělení zdrojů

Další z oblastí použití dynamického programování je speciální optimalizační metoda, spadající pod separovatelné programování, které řeší některé úlohy nelineárního programování.

Úlohy nelineárního programování jsou obecně ve tvaru

$$f(x_1, \dots, x_n); g_i(x_1, \dots, x_n) \leq 0; i = 1, \dots, m \quad (5)$$

nebo

$$f(x_1, \dots, x_n); g_i(x_1, \dots, x_n) = 0; i = 1, \dots, m \quad (6)$$

kde je alespoň jedna z funkcí  $f$  nebo  $g_i$  nelineární.

Separovatelná funkce je ve tvaru

$$f(x_1, \dots, x_n) = \sum_{i=1}^n f_i(x_i) \quad (7)$$

A omezení je zadáno jako

$$\sum_{i=1}^n x_i \leq b; x_i \geq 0 \quad (8)$$

Při řešení případů tohoto typu se začíná jednou separovanou funkcí a postupně se hledá extrém pro 2,3,4,... separované funkce, dokud nepokryjeme celou účelovou funkci. Tyto mezivýsledky se zapisují do tabulky a po dokončení první části se zpětně odečtou optimální hodnoty, pro které bude mít účelová funkce minimální/maximální hodnotu. Opět se vychází z principu optimality, kdy máme optimalizované například 3 separované funkce, a pak, když optimalizujeme čtvrtou, použijeme předchozí výsledky a dopočítáváme pouze hodnoty aktuální funkce. [9]

Celý algoritmus optimalizace [10]:

1. Ekvidistantně rozdělit  $b$  s krokem  $h$  na  $b_k$
2. Výpočet hodnot  $F_1(x_1)=f_1(x_1)$
3. Optimalizace  $F_2(x_2)=extr(0 \leq x_2 < b)\{f_2(x_2)+F_1(b-x_2)\}$
4. Optimalizace  $F_3(x_3)=extr(0 \leq x_3 < b)\{f_3(x_3)+F_2(b-x_3)\}$
- ...
- n+1. Optimalizace  $F_n(x_n)=extr(0 \leq x_n < b)\{f_n(x_n)+F_{n-1}(b-x_n)\}$
- n+2. Zpětné odečtení optimálního řešení  $x_n, x_{n-1}, x_{n-2}, \dots, x_1$ .

### 2.4.1 Příklad

Je zadána účelová funkce

$$f(x_1, x_2, x_3) = 3 * x_1^2 + 2 * x_2^2 + x_3^2$$

a omezující podmínky

$$x_1 + x_2 + x_3 = 1; x_1, x_2, x_3 \geq 0$$

Krok  $h=0,2$ . Budeme hledat minimum této funkce.

Sestavíme tabulku, do které budeme postupně vepisovat získané hodnoty. První sloupec  $b_k$  udává použitelné hodnoty podle zadaného omezení a kroku.

Následují dvojsloupece pro každou separovanou funkci. Hodnoty pro první dvojsloupec pro  $x_1(b)$  a  $F_1$  se získají přímým dosazením  $b_k$  do funkce  $f_1 (3 * x_1^2)$ . Pro další dvojsloupece je již postup složitější. Hodnota  $b_k$  pro daný řádek nám udává maximální hodnotu, kterou můžeme rozdělit mezi  $x_1$  a  $x_2$  ( $x_1 + x_2 = b_k$ ). Například  $b_k 0,2$  můžeme rozdělit jako  $x_1=0$  a  $x_2=0,2$  nebo  $x_1=0,2$  a  $x_2=0$ . Pro obě tyto kombinace vypočítáme hodnotu  $F_2 (2 * x_2^2 + F_1(x_1))$  a vybereme tu, která je nejmenší (pokud hledáme minimum). Přitom znovu nepočítáme hodnotu  $F_1$ , ale tuto hodnotu odečteme z příslušného řádku v tabulce. Stejným způsobem pokračujeme pro další hodnoty  $b_k$ . U funkce  $F_3$  nám v tomto případě stačí vypočítat hodnotu pro  $b_k=1$ , protože v omezující podmínce je zadáno, že součet všech hodnot se rovná jedné. Pokud by v omezující podmínce bylo  $\leq$ , pak by se počítala hodnota  $F_3$  pro všechny hodnoty  $b_k$ .

b <sub>k</sub>	x <sub>1</sub> (b)	F <sub>1</sub>	x <sub>2</sub> (b)	F <sub>2</sub>	x <sub>3</sub> (b)	F <sub>3</sub>
0	0	0	0	0	-	-
0,2	0,2	0,12	0,2	0,08	-	-
0,4	0,4	0,48	0,2	0,2	-	-
0,6	0,6	1,08	0,4	0,44	-	-
0,8	0,8	1,92	0,4	0,8	-	-
1	1	3	0,6	1,2	0,6	0,56

Tab. 1. Tabulka s výpočty příkladu dělení zdrojů

Příklad výpočtu  $F_2$  pro  $b_k=0,4$ ;  $x_1+x_2=b_k$

$$(x_1, x_2) = F_1(x_1) + f_2(x_2) = F_2(b_k)$$

$$(0,4; 0) = 0,48 + 0 = 0,48$$

$$(0,2; 0,2) = 0,12 + 0,08 = \underline{0,2} \text{ - minimum}$$

$$(0; 0,4) = 0 + 0,32 = 0,32$$

Po dokončení tabulky provedeme odečtení výsledných hodnot. Začneme hodnotou  $x_3$  – ta má mít pro dosažení minima účelové funkce podle tabulky hodnotu 0,6. Celkové omezení pro součet  $x_1 + x_2 + x_3$  je 1, pro součet  $x_1 + x_2$  nám tedy zbývá 0,4. Odečteme tedy z tabulky hodnotu  $x_2$  pro  $b_k = 0,4$  a zjistíme, že optimální hodnota  $x_2$  je 0,2. Na  $x_1$  zbývá 0,2.

Zpětný chod:

$$x_3 = 0,6$$

$$x_2 = 0,2$$

$$x_1 = 0,2$$

#### 2.4.2 Příklad se substitucí

Může dojít k situaci, že omezení není zadáno ve tvaru

$$\sum_{i=1}^n x_i \leq b; x_i \geq 0 \quad (9)$$

ale

$$\sum_{i=1}^n a_i x_i \leq b; x_i \geq 0 \quad (10)$$

kde  $a_i$  je koeficient násobící hodnotu dané proměnné.

Při takovém zadání je potřeba provést substituci, která omezující podmínku převede do požadovaného tvaru

$$y_i = a_i x_i \quad (11)$$

Tuto substituci musíme promítnout i do účelové funkce

Celý výpočet provedeme se substituovanými proměnnými  $y_i$  a po odečtení výsledku ze zpětného chodu se provede zpětná substituce.

Pro ukázkou použijeme stejnou účelovou funkci jako v předchozím případě, pouze upravíme omezující podmínku.

Je zadána účelová funkce

$$f(x_1, x_2, x_3) = 3 * x_1^2 + 2 * x_2^2 + x_3^2$$

a omezující podmínky

$$x_1 + 2 * x_2 + x_3 = 1; x_1, x_2, x_3 \geq 0$$

Krok  $h=0,2$ . Budeme hledat minimum této funkce.

Než začneme sestavovat tabulku, musíme provést substituci pro dosažení požadovaného tvaru omezující podmínky.

Substituce tedy bude vypadat následovně:

$$y_1 = x_1$$

$$y_2 = 2 * x_2; x_2 = \frac{y_2}{2}$$

$$y_3 = x_3$$

$$y_1 + y_2 + y_3 = 1$$

$$f(y_1, y_2, y_3) = 3 * y_1^2 + 0,5 * y_2^2 + y_3^2$$

Sestavíme tabulku a vypočítáme hodnoty substituované funkce

bk	y1(b)	F1	y2(b)	F2	y3(b)	F3
0	0	0	0	0	-	-
0,2	0,2	0,12	0,2	0,02	-	-
0,4	0,4	0,48	0,4	0,08	-	-
0,6	0,6	1,08	0,6	0,18	-	-
0,8	0,8	1,92	0,6	0,3	-	-
1	1	3	0,8	0,44	0,2	0,34

Tab. 2. Tabulka s výpočty příkladu dělení zdrojů se substitucí

Při zpětném chodu odečteme hodnoty  $y_1$ ,  $y_2$  a  $y_3$

$$y_1=0,2$$

$$y_2=0,6$$

$$y_3=0,2$$

Provedeme zpětnou substituci a získáme výsledné hodnoty  $x_1$ ,  $x_2$  a  $x_3$ .

$$x_1=0,2$$

$$x_2=y_2/2=0,3$$

$$x_3=0,2$$

### 2.4.3 Existující aplikace řešící problém dělení zdrojů

Řešení problému dynamického programování a dělení zdrojů se věnuje nespočet vědeckých článků a knih, málokdo však řešení tohoto problému implementoval v podobě programu, který pro obecné zadání vypočítá řešení. Webovou aplikaci se nepodařilo nalézt žádnou.

Tomuto problému se věnoval Michal Hájek ve své bakalářské práci [2]. Ten algoritmus implementoval v podobě funkce v prostředí Matlab. Jeho implementace počítá pouze s možností násobení hodnoty proměnné koeficientem a umocnění. Složitější funkce jako odmocnina, logaritmus, sinus nebo zadávání konstant jeho implementace nepodporuje. Stejný problém řešil i Bc. Bohumil Gajda v jeho diplomové práci [9]. Ten algoritmus naprogramoval v prostředí Mathematica. Jeho řešení je navrženo přesně pro tři separované funkce a není univerzální. Jednotlivé funkce, hodnotu omezení i krok totiž zadává přímo do zdrojového kódu, nikoliv jako parametry volané funkce. Pro obecné použití je tedy toto řešení špatně použitelné. Navíc v příkladu uvedeném u algoritmu dospěl ke špatnému výsledku. Jelikož program Mathematica není volně dostupný, nemohu jeho výsledek ověřit, jestli je chybná implementace algoritmu nebo je pouze špatně uvedený výsledek. V zadání chybného příkladu se hledá minimum účelové funkce

$$f(x_1, x_2, x_3) = \sin(2x_1) + \tan(x_2) + \frac{4}{\pi}x_3$$

s omezením

$$x_1 + x_2 + x_3 = \frac{\pi}{4}; x_1, x_2, x_3 \geq 0$$

a krokem

$$h = \frac{\pi}{40}$$

Dospěl k výsledku

$$x_1 = \frac{\pi}{10}; x_2 = 0; x_3 = \frac{3\pi}{20}$$

a výsledné hodnotě účelové funkce 6,2467. Přitom dosazením výsledných hodnot do účelové funkce dostaneme hodnotu 1,188. I bez použití jakékoliv metody optimalizace nalezneme menší hodnotu dosazením  $\frac{\pi}{4}$  do funkce tangens a získáme výsledek 1.

### 3 WEBOVÉ APLIKACE

World Wide Web, často zkracovaný jako WWW nebo jen Web, je soustava propojených dokumentů přenášených pomocí protokolu HTTP (Hyper Text Transfer Protokol). Tyto dokumenty jsou nejčastěji ve formátu HTML (Hyper Text Markup Language – značkovací jazyk). Protokol HTTP patří do aplikační vrstvy referenčního modelu ISO/OSI, který popisuje principy a standardy komunikace v počítačových sítích. [11]

Aplikace je softwarové vybavení počítače, který slouží uživateli k vykonávání užitečných činností nebo řešení problémů.

Webová aplikace pak je tedy program, který pomáhá uživateli řešit úkoly, je realizován na webovém serveru a klient k němu přistupuje přes počítačovou síť. Jako klient slouží webový prohlížeč, který zobrazuje dokumenty generované na serveru.

Klientský počítač tak neobsahuje žádné části aplikace a slouží jen k odesílání požadavků a zobrazování odpovědí. Celá aplikace je realizována na serveru.

Předchůdce webových aplikací byly klasické aplikace architektury klient-server, které měly 2 části – klientskou a serverovou. Ty nepoužívají k přenosu dat protokol http, ale vlastní protokol navržený přímo pro přenos požadovaných dat a informací. Není tak obecný, ale díky tomu může být jednodušší. Tyto aplikace jsou také stavové – tj klient i server si pamatují předchozí kroky a požadavky a posílají pouze potřebné údaje. Klient tak nemusí pokaždé posílat přihlašovací údaje a celé znění požadavku. Naproti tomu bezstavový protokol HTTP neumí uchovávat stav komunikace a jednotlivé dotazy spolu nesouvisí. Tato vlastnost je velmi nepříjemná při implementaci složitějších systémů a obchází se pomocí HTTP Cookies [12] nebo AJAXu.

Webová aplikace už z principu nemůže být pouhá soustava statických stránek. Statické webové stránky jsou uloženy na serveru v souborech a při obdržení požadavku na jejich zobrazení se klientovy pošlou tak, jak jsou. Tímto způsobem nelze zajistit žádnou interaktivitu. Z toho důvodu musí být webová aplikace realizována jako dynamická – mění svůj obsah na základě parametrů, které od uživatele obdrží. Toto generování nebo změna obsahu může být provedena na straně serveru, i na straně klienta.

V případě vytváření obsahu na straně klienta server zašle v rámci webové stránky i kód, který reaguje na zadání uživatele a samotné generování obsahu probíhá uvnitř webového

prohlížeče a server se o obsah po prvotním odeslání stránky s kódem dále nezajímá. K těmto účelům se nejčastěji používá JavaScript nebo Flash.

Při tvorbě obsahu na straně serveru se veškerý kód vykoná před odesláním odpovědi a klient nezjistí, jakým způsobem byl výsledek vygenerován (u JavaScriptu obdrží čitelný zdrojový kód). Server může odpověď generovat podle času, obsahu databáze nebo parametrech URL adresy POST a GET. Metody POST a GET slouží k předání dat z formuláře, které jsou ve tvaru parametr=hodnota. Na jejich obsah pak server zareaguje a vygeneruje příslušnou odpověď.

Tyto přístupy je možné libovolně kombinovat podle požadovaného výsledku a některý obsah generovat na serveru a zbytek na straně klienta.

Výhody generování na straně serveru jsou především skrytí zdrojového kódu, které pomáhá ke zvýšení bezpečnosti a také zabraňuje kopírování. Nevýhoda pak spočívá ve vyšších nárocích na výkon serveru, kdy v případě složitých výpočtů může být server přetížen.

Na straně serveru se o generování obsahu stará serverový software, který je implementován buďto jako zkompileovaná běžící aplikace, nebo jako interpret zdrojového kódu uloženého v souborech. Dnes se k tomuto účelu nejčastěji používá PHP, ASP, Java, Perl nebo Python.

### 3.1 Apache

Apache HTTP Server je multiplatformní softwarový webový server vyvíjený jako OpenSource software. Umožňuje spouštění skriptů napsaných v několika jazycích, nejčastěji je ovšem používán jazyk PHP. PHP je rekurzivní zkratka pro PHP: Hypertext Preprocessor (dříve se používalo Personal Home Page tools). Jeho syntaxe je odvozena od jazyků C, Java a Perl. Hlavním zaměřením jazyka je na rychlý vývoj dynamicky generovaných stránek. [13]

PHP umožňuje procedurální i objektově orientované programování. Dále PHP nabízí pokročilé funkce pro zpracování textu (od regulárních výrazů až po parsování XML dokumentů). Podle statistik se jedná o nejvyužívanější platformu již od roku 1996, kdy se PHP dostalo před NCSA HTTPd. Vrcholná léta PHP byla v letech 2000 až 2007, kdy tržní podíl serveru Apache přesahoval 60% podílu na trhu. [14]

### 3.2 Internet Information Services

Internet Information Services (zkráceně IIS) je softwarový webový server firmy Microsoft. Podporuje několik protokolů a programovacích jazyků. Microsoft ho vyvíjí především pro své serverové systémy Microsoft Server, ale je možné ho provozovat i na desktopových operačních systémech této firmy. IIS umožňuje běh skriptovacích enginů ASP a ASP.NET. Active Server Pages je objektově založené, tj. obsahuje několik základních tříd, ale není objektově orientované, takže nelze další třídy definovat. Tato platforma již není dále rozvíjena, protože se vývoj přesunul na objektově orientovanou verzi ASP.NET, ale je nadále podporována i ve Windows 8 a bude podporována ještě příštích 9 let (tj do roku 2022) [15]. ASP.NET je objektově orientovaný skriptovací engine. Zdrojový kód se na rozdíl od ASP neinterpretuje, ale automaticky kompiluje. Tato kompilace probíhá ve dvou etapách. Při prvním požadavku na stránku se zdrojový kód zkompiluje do jazyka MSIL a ten je pak Just In Time kompilován až při zpracování požadavku od klienta.

V ASP je výchozí programovací jazyk VBScript, kromě něho je podporován JavaScript. Dále je možné doinstalovat podporu pro PERL nebo Python. V ASP.NET jsou standardně podporovány jazyky VisualBasic.NET, JScript.NET, C#, J# a další jazyky podporující CLI. [16]

### 3.3 Java

Pojmenování Java se používá jak pro označení platformy umožňující spouštění aplikací, tak i pro konkrétní programovací jazyk. Tuto platformu je možné provozovat na široké škále zařízení, od mobilních telefonů až po superpočítače. Pro běh webového serveru se používá její součást Java Enterprise Edition (Java EE) a její rozšíření Java Servlets a Java Server Pages. O vývoj systému se původně staral Sun, který ho později uvolnil jako Open source a v letech 2009-2010 vývoj přebíral Oracle. [17]

Programovací jazyk Java objektově orientovaný. Napsaný zdrojový kód se zkompiluje do interního jazyka platformy Java – tzv. bytecode. Tento bytecode je pak Just In Time kompilován v prostředí Java Virtual Machine. Toto virtuální prostředí je vytvořené pro všechny běžné operační systémy. Díky tomu lze jednou zkompilovaný bytecode spouštět na různých platformách.

## 3.4 Bezpečnost

Zabezpečení je dnes základní pilíř každého síťového zařízení nebo aplikace. Existuje nespočet nebezpečí, která mohou ohrozit provoz webové aplikace. Útočník může napadnout základní síťovou infrastrukturu pomocí chyb ve firmware routerů, pokusit se přeměrovat požadavky z cílového serveru na svůj podvržený, přetížit server velkým množstvím požadavků, odposlouchávat komunikaci klientů a serveru, napadnout operační systém serveru, napadnout aplikaci mající na starost provoz webových stránek, napadnout libovolnou jinou aplikaci běžící na serveru, která je zranitelná, napadnout webovou aplikaci samotnou, nebo třeba i provést fyzický útok na umístění serveru. Drtivou většinu těchto nebezpečí není možné ošetřit na úrovni aplikace, jsou to totiž útoky na nižší vrstvy ISO/OSI modelu. Blíže se podíváme jen na problémy, které je možné ovlivnit při implementaci webové aplikace.

### 3.4.1 Injection

Injection, nebo také injektování kódu, je podstrčení škodlivého kódu pomocí neošetřeného formuláře, podvržené cookie nebo manipulací s URL adresou. Tento typ útoku je nebezpečný zejména pro systémy, které používají dva a více různých programů a ty si předávají požadavky a data. Nejčastějším případem tohoto typu útoku je SQL Injection, tedy proniknutí do databáze napadené aplikace.

Budeme například uvažovat redakční systém, který na základě ID článku v URL adrese zobrazuje jednotlivé články. Tato funkce by mohla být implementována následovně:

```
1 $SQL_Dotaz="Select * from clanky where id=$_GET["id"]"
```

Jenže útočník může podobný zápis předvídat a záměrně do URL adresy místo pouhého ID článku zadá řetězec "1 and 1=1". Dotaz po obdržení tohoto kódu bude mít podobu:

```
1 Select * from clanky where id=1 and 1=1
```

Tato podmínka bude vždy splněna a zobrazí se všechny články v databázi. Útočník tím pozná, že objevil bezpečnostní slabinu aplikace a může vymýšlet, jak provozovatele aplikace poškodit. Může například smazat všechny články uložené v databázi pomocí dotazu:

```
1 Select * from clanky where id=1 and DROP Table clanky
```

Před útoky tohoto typu není těžké se bránit, jen je potřeba ověřit každý uživatelský vstup, zda odpovídá očekávané podobě nebo neobsahuje nežádoucí znaky či řetězce. [18]

### 3.4.2 XSS - Cross-site scripting

Cross-site scripting je útok na neošetřené formulářové pole. V předchozím případě se útočník pokoušel napadnout samotnou aplikaci. Útok pomocí XSS je veden na JavaScript a může být zneužit k poškození obsahu stránky nebo k získání důvěrných údajů od uživatelů. Tento typ útoku opět pracuje s předpokladem, že vstupy formuláře nebo parametry URL adresy nejsou ošetřené. Útočník modifikuje URL tak, aby byl do zdrojového kódu stránky vložen jeho vlastní JavaScriptový kód. Tuto URL odešle uživatelům, kteří kliknou na odkaz na důvěryhodnou webovou stránku, ale při jejím navštívení se spustí útočnickův kód a buďto se útočník pokusí získat od uživatele jménem provozovatele stránky citlivé údaje, nebo podvrhne na stránku falešný obsah. [19]

V praxi pak tento útok vypadá třeba následovně:

Ve zdrojovém kódu je zobrazeno uživatelem zadané jméno

```
1 <?php echo $_GET['jmeno']; ?>
```

Útočník do formuláře místo jména zadá

```
1 "Josef<script>alert('Toto je XSS útok')</script>"
```

Po načtení stránky zobrazí útočnickovo hlášení. To samo o sobě není nebezpečné, dokud útočník nepošle odkaz uživatelům důvěřujícím napadené stránce. Napadené URL by vypadalo takto:

```
www.znamastranka.cz/index.php?jmeno=Josef<script>alert('Toto je XSS útok')</script>
```

Z pohledu webové aplikace implementující matematický algoritmus, která neukládá žádná data, jsou tak nebezpečné zejména útoky typu XSS.

## **II. PRAKTICKÁ ČÁST**

## 4 NÁVRH ŘEŠENÍ PROBLÉMU DĚLENÍ ZDROJŮ

Cílem této práce je implementace aplikace, která řeší tabulkovou formou úlohy dynamického programování pomocí principu dělení zdrojů. Tato aplikace tedy musí od uživatele získat údaje potřebné pro výpočet – typ hledaného extrému, účelovou funkci, omezující podmínku a velikost kroku. Je tedy potřeba zobrazit uživateli formulář, do kterého tyto údaje zadá a pokud bude jeho vstup korektní, zobrazit mu výsledek. Výpis výsledku je požadovaný třemi různými způsoby – jednorázově, krokovat výpočet nebo výsledek uložit jako soubor PDF.

### 4.1 Princip algoritmu

Základní princip fungování aplikace je znázorněn na obrázku 6.



Obr. 6. Schéma fungování aplikace

Získání údajů od klienta je u HTTP možné pouze pomocí metody POST nebo GET – tj odeslání údajů zadaných do formuláře v URL (GET) nebo v těle HTTP požadavku.

Kontrola vstupu je důležitá část zejména z hlediska bezpečnosti – na správně nakonfigurovaném serveru je to jediná možnost, jak napadnout aplikaci (tedy z těch možností, které je možné v rámci implementace aplikace ovlivnit). Každý vstup je nutné zkontrolovat ještě před jeho prvním použitím.

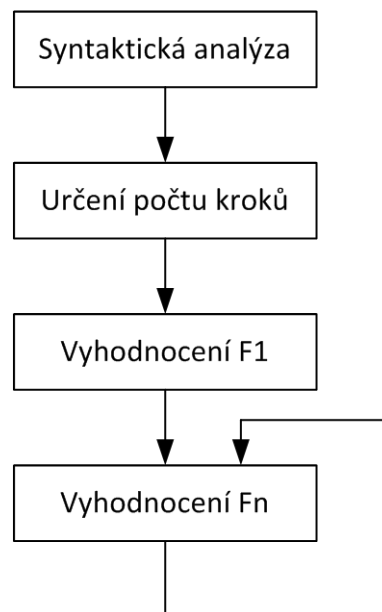
Samotný výpočet je nutné rozdělit do několika kroků. Ty jsou znázorněny na obrázku 7. První z nich je syntaktická analýza, nebo také parsování zadaných vstupů a samotné

zjištění významu textu zadaného uživatelem. Účelová funkce může mít různé podoby a používat mnoho matematických funkcí a není uživatelsky přívětivé vytvořit rozsáhlý a složitý formulář, který by umožňoval její zadání výběrem konkrétních funkcí a hodnot v předem připravených polích. Ani by nebylo možné vytvořit dostatečně univerzální formulář, který by uživatele neomezoval. Tato funkce musí být zadána ve formě textového řetězce a ten syntaktickou analýzou zpracovat a vyhodnotit ho. Obdobným způsobem bude zpracována i omezující podmínka.

V případě, že omezující podmínka obsahuje koeficienty násobící hodnoty jednotlivých proměnných, je nutné provést substituci. Její provedení bez použití sofistikovaného systému specializovaného na matematické operace přesahuje rozsah této diplomové práce. Účelová funkce nebude převedena do tvaru po substituci, ale substituci lze provést jednoduše tak, že se pro výpočet hodnot jednotlivých separovaných funkcí použije vstupní hodnota upravená podle potřebné substituce jednotlivých proměnných.

Pro výpočet je nezbytné rozdělení intervalu  $\langle 0, b \rangle$  (kde  $b$  je hodnota na pravé straně rovnice omezující podmínky) na množinu hodnot  $b_k$  podle zadaného kroku  $h$ . Při implementaci je nutné ošetřit situaci, že  $h$  nedělí  $b$  beze zbytku.

Výpočet hodnot funkcí pak probíhá nejprve pro  $F_1$  samostatně, pro  $F_2, \dots, F_n$  v kombinaci s vyhledáním hodnoty předchozí funkce. Z vypočítaných hodnot se hledá minimum nebo maximum.



Obr. 7. Postup výpočtu

Na závěr jsou vypsány vypočítané výsledky. V případě zobrazení konečného výsledku a krokování řešení půjde o zobrazení dat stejnou formou. Je tedy vhodné je implementovat jako jednu funkci a k zobrazení kompletního řešení přistupovat jako k poslednímu kroku postupného výpisu.

Výstup do PDF není standardní funkce nebo součást žádné z běžných platforem – PHP, ASP ani Java, tuto funkci je tedy nutné implementovat pomocí externí služby, knihovny nebo funkce.

## 4.2 Volba platformy

Výše popsáný návrh algoritmu neobsahuje žádné specifické funkce, které by byly dostupné pouze na jedné z představených platforem. Jelikož je hlavní cíl implementace algoritmu, a ten nepracuje s žádnými složitými datovými strukturami, maximálně s dvourozměrným polem, není potřeba objektově orientované řešení – ani v tomto případě neexistují vhodné objekty a vlastnosti, které by OOP přístup umožňovaly. Implementace algoritmu vyžaduje procedurální přístup a jeho implementace pomocí OOP by byla vynucená.

Na Ústavu řízení procesů je provozován MAT Server, na kterém jsou provozovány některé studentské projekty, a je na provozován na Microsoft IIS. Z toho důvodu byla pro implementaci zvolena platforma ASP. Výslednou aplikaci tak bude možné umístit na MAT Server a neměla by představovat žádné omezení.

## 5 POPIS IMPLEMENTACE WEBOVÉ APLIKACE

Implementace webové aplikace byla provedena podle návrhu algoritmu v předcházející kapitole. Aplikace je založena na platformě ASP a je tedy možné ji provozovat na Microsoft IIS Serveru, který je zdarma dostupný v desktopových operačních systémech Microsoft Windows nebo v serverových systémech Microsoft Server.

### 5.1 Interní struktura aplikace

Implementace aplikace má rozsah přes 700 řádků zdrojového kódu. Kvůli lepší správě a přehlednosti jsou jednotlivé části rozděleny podle funkce do více souborů. Struktura zdrojových souborů je následující:

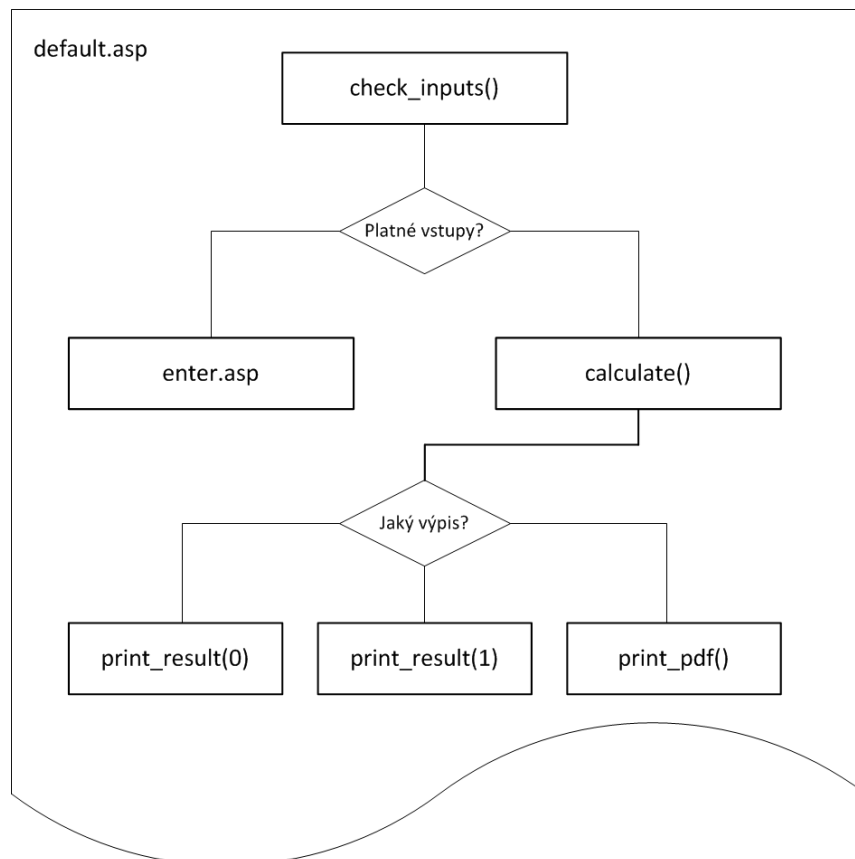
Název souboru	Popis
<i>default.asp</i>	Úvodní stránka
<i>/files/common.asp</i>	Knihovna funkcí realizujících samotný výpočet řešení
<i>/files/enter.asp</i>	Stránka obsahující formulář pro zadání úlohy a nápovědu
<i>/files/fpdf.asp</i>	ASP implementace knihovny FPDF umožňující výstup do PDF
<i>/files/pdf.asp</i>	Funkce sloužící k výpisu výsledku do PDF souboru
<i>/files/print.asp</i>	Funkce zajišťující výpis výsledku (krokováného i jednorázového)
<i>/files/step.asp</i>	Formulář se skrytými vstupy a tlačítkem pro krokování výpisu
<i>/files/styles.css</i>	CSS styly s použitým formátováním
<i>/files/fpdf/</i>	Složka s pomocnými soubory FPDF knihovny (4 složky, 22 souborů)

*Tab. 3. Struktura souborů v implementované aplikaci*

Všechny kroky ze schéma fungování aplikace z minulé kapitoly jsou implementovány jako samostatné funkce, s výjimkou zobrazení výsledku a krokování řešení, které jsou implementovány jako jedna funkce ovlivněná zadaným parametrem.

Název výchozí automaticky zobrazované ASP stránky na IIS serveru je *default.asp*, z toho důvodu je právě takto pojmenována úvodní stránka. Ta napřed naimportuje obsah souboru

*common.asp*, nastaví výchozí hodnoty a ověří vstupy zadané uživatelem. Podle uživatelského vstupu je zobrazen buďto formulář sloužící pro zadání úlohy, chyba informující o špatném zadání, výpis řešení, krokování řešení, nebo PDF soubor. Jak je vidět na obrázku 8, tato výchozí stránka zajišťuje provedení všech částí, ze kterých se aplikace skládá.



Obr. 8. Vývojový diagram aplikace

### 5.1.1 Získání údajů

Pokud je stránka poprvé navštívena, je zobrazen formulář pro zadání údajů ze souboru *enter.asp*. Ten neobsahuje žádné zvláštní funkce, kromě zobrazování a skrývání nápovědy pomocí JavaScriptu. Podrobný popis formuláře se nachází v kapitole 5.2.

### 5.1.2 Kontrola vstupu

Kontrola korektních vstupů je stěžejní část z pohledu bezpečnosti aplikace. V této části je potřeba především zabránit použití speciálních znaků, které se nebudou vyskytovat v zápisech funkcí a mohly by vést k injekci kódu nebo XSS útoku.

Kontrola probíhá ihned jako první. Přijaté řetězce jsou zkontrolovány na přítomnost nežádoucích znaků. Přesněji řečeno, jsou řetězce kontrolovány, zda obsahují pouze povolené znaky pro dané vstupní pole. Pro každé vstupní pole je sestaven regulární výraz, který zjistí přítomnost pouze povolených znaků. V případě zadání nepovoleného znaku se daný řetězec dále nezpracovává a jeho hodnota je zahozena. Žádný výpočet se neprovede a je znovu zobrazen vstupní formulář, který obsahuje ty zadané údaje, které validaci vyhověly. Není nutné tak znova zadávat celé zadání znovu.

Regulární výrazy kontrolující jednotlivá vstupní pole jsou:

- Účelová funkce: `"^[A-z0-9\ (\) \^\/\ *-\+\. \, ]+$"` – písmena, čísla a matematické operátory.
- Výběr hledání minima/maxima: `"^[minax]+$"` – pouze písmena ze slov min a max.
- Levá strana rovnice omezení: `"^[A-z0-9\ *\+\. ]+$"` – pouze písmena, čísla, +, \* a tečka.
- Operátor rovnice omezení: `"^[<=]+$"` – pouze znaky = nebo <.
- Pravá strana rovnice omezení: `"^[0-9\ . ]+$"` – pouze čísla a tečka.
- Krok h: `"^[0-9\ . ]+$"` – pouze čísla a tečka.
- Krok výpisu řešení: `"^[0-9]+$"` – pouze čísla.
- Typ výpisu řešení: `"^[steprultdf]*$"` – pouze písmena ze slov step, result a pdf.

Aplikace tedy nepředpokládá, že vždy obdrží data pouze od jejího formuláře, ale kontroluje i údaje, které uživatel běžně nemůže změnit. Tento přístup přispívá k zabezpečení aplikace a nedůvěřuje žádnému údaji, který od uživatele obdrží.

### 5.1.3 Výpočet řešení

Výpočet řešení je realizován pomocí funkce *calculate()*. Ta jako první volá funkce zpracovávající zadání účelové funkce a omezující podmínku. Poté je určen počet kroků, voláno vyhodnocení funkce  $F_l$  a vyhodnocení dalších funkcí.

Jako první je zpracována podmínka, protože její zápis je jednodušší (pouze násobení koeficientem a sčítání proměnných) než zápis účelové funkce obsahující nelineární, goniometrické a další funkce. O toto zpracování se stará funkce *split\_lim(fce)*. Ta funkci, kterou obdrží v argumentu, rozdělí podle znamének + na jednotlivé části. Ty pak ještě

kontroluje na přítomnost koeficientu násobícího danou proměnou (které značí provedení substituce). Také určí názvy použitých proměnných, zda použité proměnné nejsou shodné s klíčovými slovy jazyka VBScript nebo s názvy funkcí a proměnných v aplikaci.

Funkce *split\_fce(fce)* zpracuje účelovou funkci. Rozdělí ji na jednotlivé části podle znamének +, které se nenacházejí uvnitř závorek. Také zkontroluje, zda účelová funkce obsahuje stejné proměnné jako omezující podmínka, zda jsou funkce v účelové funkci separované a zda stejně jako omezení neobsahuje klíčová slova nebo názvy funkcí a proměnných.

O samotné vyhodnocení separovaných funkcí se stará funkce *eval\_expr(expr, val)*. Pro separovanou funkci a hodnotu proměnné, které obdrží v argumentu, určí hodnotu. Ta je určena pomocí funkce jazyka VBScript *eval()*, která vykoná výraz zadaná jako argument, jako kdyby se jednalo o zdrojový kód. Díky této funkci tedy není třeba provádět syntaktickou analýzu, neboť zpracuje funkci přímo v textové podobě, ale naopak je tato funkce velmi nebezpečná z hlediska zabezpečení aplikace. Vykonání kódu zadaného uživatelem znamená velké bezpečnostní riziko, proto jsou uživatelské vstupy kontrolovány regulárním výrazem a také na přítomnost klíčových slov jazyka a názvy funkcí a proměnných. Při nedodržení těchto kontrol by byla aplikace velmi snadno zranitelná.

O výpočet hodnot funkcí  $F_2$  až  $F_n$  se stará funkce *eval\_bk(expr, bk, col, print)*. Ta pro zadanou separovanou funkci a hodnotu  $b_k$  volá funkci *eval\_expr()*, přičítá odpovídající hodnotu z minulého sloupce a hledá minimum nebo maximum z vypočítaných hodnot.

#### 5.1.4 Výpis řešení

Výpis je realizován dvěma způsoby. Jeden je společný pro přímý výpis řešení a krokování výpisu, druhý způsob je uložení výsledku do souboru PDF.

Celý výpis řešení je realizován pomocí procedury *print\_result(print\_step)*. Pokud má argument hodnotu nula, je vypsáno celé řešení. V případě, že se postupně krokuje řešení, je na stránku před samotným výpisem vložen formulář obsahující zadání ve skrytých polích a tlačítko Další krok, které formulář odešle se stejným zadáním, jen zvýšeným krokem výpisu.

Uložení do souboru PDF není standardní součástí jazyka VBScript, takže bylo nutné použít externí knihovnu nebo funkci. Funkcí vhodných pro použití v ASP.NET existuje několik, pro ASP pouze jedna - ASP FPDF [20]. Jedná se o ASP implementaci open source

knihovny FPDF, která byla původně vyvinuta pro PHP. Tato knihovna je snadno použitelná, jen má problémy s podporou české diakritiky. Jedná se o omezení na straně této knihovny, proto ve výsledných PDF není použita diakritika. Pokud účelová funkce obsahuje více než 5 proměnných, je orientace stránky změněna na šířku.

Použití knihovny ASP FPDF není založeno na uložení obsahu vygenerované stránky do souboru PDF, ale vyžaduje výpis všech textů pomocí vlastní funkce. Celý výstup do PDF je tedy samostatně implementován stejným způsobem jako výpis řešení, ale místo volání funkce *Response.Write("Text")* je volána fpdf funkce *pdf.Cell šířka,výška,"Text"*.

## 5.2 Uživatelské rozhraní

Aplikace komunikuje s uživatelem pomocí jednoduchého uživatelského rozhraní. Má totiž prakticky jen dva úkoly – získat od uživatele zadání úlohy a zobrazit mu vypočítaný výsledek.

Pro získání zadání řešené úlohy slouží uživateli formulář s několika vstupními poli (obrázek 9). Ty jsou rozděleny do bloků podle toho, jestli se týkají zadání účelové funkce, omezující podmínky, kroku výpočtu, nebo volby výpisu řešení.



Obr. 9. Úvodní strana aplikace

Pro zadání celé účelové funkce slouží textové pole. Do něho se zapíše funkce v textovém tvaru. Tento zápis není na první pohled příliš pohodlný, a pro zápis jednoduchých funkcí

toto řešení nemusí dávat smysl. Pokud by zadání funkce bylo řešeno pomocí několika výběrových polí (např. výběr funkce pro každou proměnnou), bylo by zadávání jednoduchých funkcí snazší. Hlavní výhoda textového zápisu spočívá v možnosti zadání složitých funkcí, které mohou být různě kombinované a vnořené. Jako příklad jednoduché funkce lze uvést  $2 \cdot x_1^2$  nebo  $\sqrt{x_2}$ . Složitější zadání pak může být například  $x_3 \cdot \log_{10}((x_3+2)^2)$ . Zadání takové funkce nelze jednoduše realizovat pomocí pevně dané struktury zadávání funkce.

Výběr, zda chceme hledat minimum nebo maximum účelové funkce, je realizován přepínačem.

Zadání omezující podmínky je rozděleno do tří částí. Zadání levé části rovnice je ve formě textového pole, výběr, zda se jedná o  $=$  nebo  $\leq$ , má podobu výběrového pole a zadání hodnoty pravé strany rovnice je opět textové pole.

Zadání kroku pro výpočet řešení je textové pole.

Pod zadáním účelové funkce, omezující podmínky a kroku se nachází přepínače pro volbu typu výpisu řešení úlohy a tlačítko OK, které odešle zadání serveru a spustí výpočet.

Pod formulářem pro zadání řešené úlohy je nápověda k aplikaci, dynamickému programování a vzorové příklady. Tyto tři části jsou standardně skryty a dají se zobrazit pomocí kliknutí na příslušný odkaz.

Do všech polí je možné zadat desetinná čísla pouze ve tvaru s desetinnou tečkou, nikoliv s desetinnou čárkou.

Pokud je do formuláře zadáno platné zadání úlohy, zobrazí se po kliknutí na tlačítko OK výsledek v požadované formě.

Výpis kompletního řešení obsahuje vypsání zadání úlohy, substituci proměnných (pokud je třeba), tabulku obsahující hodnoty všech funkcí  $F_x$  a zpětný chod i s případnou zpětnou substitucí. Na konec je vypsána minimální nebo maximální hodnota účelové funkce po dosažení nalezeného optimálního řešení. Tento výpis je zobrazen na obrázku 10.

Minimum účelové funkce:  $40 \cdot x_1^2 + x_2^2 + 10 \cdot x_3^2$   
 Omezující podmínka:  $2 \cdot x_1 + 0.2 \cdot x_2 + 0.8 \cdot x_3 = 8 \quad h=1$   
 Provedena substituce:  
 $y_1 = 2 \cdot x_1$   
 $y_2 = 0.2 \cdot x_2$   
 $y_3 = 0.8 \cdot x_3$

bk	y1(b)	F1	y2(b)	F2	y3(b)	F3
0	0	0	0	0	-	-
1	1	10	0	10	-	-
2	2	40	1	35	-	-
3	3	90	1	65	-	-
4	4	160	1	115	-	-
5	5	250	1	185	-	-
6	6	360	2	260	-	-
7	7	490	2	350	-	-
8	8	640	2	460	2	322.5

Zpětný chod:  
 $y_3 = 2 \quad x_3 = 2.5$   
 $y_2 = 2 \quad x_2 = 10$   
 $y_1 = 4 \quad x_1 = 2$

Hodnota funkce  $f=322.5$

[Zpět na úvodní stránku](#)

Obr. 10. Jednorázový výpis řešení

Výpis řešení do PDF souboru má stejnou formu. Pro 5 nebo méně proměnných je zobrazeno na stránce orientované na výšku (jako na obrázku 11), pro 6 a více proměnných je orientace stránky změněna na šířku (ukázka na obrázku 12).

Minimum ucelove funkce:  $40 \cdot x_1^2 + x_2^2 + 10 \cdot x_3^2$   
 Omezující podmínka:  $2 \cdot x_1 + 0.2 \cdot x_2 + 0.8 \cdot x_3 = 8 \quad h=1$   
 Provedena substituce:  
 $y_1 = 2 \cdot x_1$   
 $y_2 = 0.2 \cdot x_2$   
 $y_3 = 0.8 \cdot x_3$

bk	y1(b)	F1	y2(b)	F2	y3(b)	F3
0	0	0	0	0	-	-
1	1	10	0	10	-	-
2	2	40	1	35	-	-
3	3	90	1	65	-	-
4	4	160	1	115	-	-
5	5	250	1	185	-	-
6	6	360	2	260	-	-
7	7	490	2	350	-	-
8	8	640	2	460	2	322.5

Zpetny chod:  
 $y_3 = 2 \quad x_3 = 2.5$   
 $y_2 = 2 \quad x_2 = 10$   
 $y_1 = 4 \quad x_1 = 2$

Hodnota funkce  $f=322.5$

Obr. 11. Ukázka PDF orientovaného na výšku

Minimum ucelove funkce:  $x_1+2^*x_2^2+3^*x_3^3+4^*x_4^4+5^*x_5^5+6^*x_6^6+7^*x_7^7$   
 Omezující podmínka:  $x_1+x_2+x_3+x_4+x_5+x_6+x_7=1$   $h=0,1$

bk	x1(b)	F1	x2(b)	F2	x3(b)	F3	x4(b)	F4	x5(b)	F5	x6(b)	F6	x7(b)	F7
0	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	0	0	0	0	0	0	0	0	-	-
0.1	0.1	0.1	0.1	0.02	<b>0.1</b>	<b>0.003</b>	0.1	0.0004	0.1	0.0001	0.1	0	-	-
0.2	0.2	0.2	0.2	0.08	0.1	0.023	<b>0.1</b>	<b>0.0034</b>	0.1	0.0005	0.1	0.0001	-	-
0.3	0.3	0.3	0.3	0.18	0.2	0.044	0.2	0.0094	0.2	0.002	0.2	0.0004	-	-
0.4	0.4	0.4	0.2	0.28	0.3	0.101	0.2	0.0294	0.2	0.005	0.2	0.0008	-	-
0.5	0.5	0.5	0.2	0.38	0.3	0.161	0.2	0.0504	0.2	0.011	0.2	0.0024	-	-
0.6	0.6	0.6	0.3	0.48	0.3	0.261	0.3	0.0764	0.3	0.0216	0.2	0.0054	-	-
0.7	0.7	0.7	0.2	0.58	0.3	0.361	0.3	0.1334	0.3	0.0416	<b>0.3</b>	<b>0.0094</b>	-	-
0.8	0.8	0.8	0.2	0.68	0.3	0.461	0.3	0.1934	0.3	0.0626	0.3	0.0154	-	-
0.9	0.9	0.9	0.2	0.78	0.3	0.561	0.4	0.2634	0.3	0.0886	0.3	0.0259	-	-
1	1	1	0.3	0.88	0.3	0.661	0.4	0.3634	0.4	0.1276	0.3	0.0459	<b>0.3</b>	<b>0.0109</b>

Zpětný chod:  
 $x_7=0,3$   
 $x_6=0,3$   
 $x_5=0,2$   
 $x_4=0,1$   
 $x_3=0,1$   
 $x_2=0$   
 $x_1=0$

Hodnota funkce  $f=0,0109$

Obr. 12. Ukázka PDF orientovaného na šířku

Výpis postupu řešení má také podobu jako výpis kompletního řešení, ale v tabulce je v prvním kroku uvedena jen hodnota pro  $F_1(0)$  a zbývající pole tabulky jsou prázdná. Nad tabulkou je tlačítko Další krok sloužící k zobrazení dalšího kroku výpočtu. Pod tabulkou se místo zpětného chodu zobrazuje pomocný výpočet, jak byla získána poslední zobrazená hodnota. Ukázka takového výpisu je na obrázku 13. Tímto způsobem lze krokovat výpočet prvních šesti hodnot pro každou separovanou funkci. Toto omezení je použito záměrně, protože pokud by se krokoval výpočet každého kroku, nebylo by prakticky možné dojít krokováním k posledním funkcím při použití většího počtu proměnných a jemného kroku. Po dokončení krokování výpočtu hodnot v tabulce je krokován i zpětný chod a je postupně zobrazován výběr výsledných hodnot.

Další krok

Maximum účelové funkce:  $(1-(x_1-0.4)^2)+(1-(x_2-0.2)^2)$   
 Omezující podmínka:  $x_1+x_2 \leq 1$   $h=0.2$

bk	x1(b)	F1	x2(b)	F2
0	0	0.84	0	1.8
0.2	0.2	0.96	0	1.92
0.4	0.4	1	0	1.96
0.6	0.6	0.96	0.2	2
0.8	0.8	0.84	-	-
1	1	0.64	-	-

bk=0.6 rozdělené mezi x1 a x2  
 $(x_1;x_2) = F_1(bk) + f_2(bk) = F_2(bk)$

$(0.6; 0) = 0.96 + 0.96 = 1.92$   
 $(0.4; 0.2) = 1 + 1 = 2$   
 $(0.2; 0.4) = 0.96 + 0.96 = 1.92$   
 $(0; 0.6) = 0.84 + 0.84 = 1.68$

max=2 pro bk=0.2

Krokování probíhá pouze pro prvních 6 hodnot  $F_x$  (princip se nemění a krokování při jemném kroku by bylo nemožné)  
[Zpět na předchozí krok](#) [Nové zadání](#)

Obr. 13. Postupný výpis řešení s pomocnými výpočty

### 5.3 Testování

HTML kód generovaný aplikací byl ověřen pomocí validátoru W3C a validací úspěšně prošel. Odpovídá tedy standardům a neměl by činit problémy v žádném webovém prohlížeči, který tyto standardy dodržuje. Aplikace byla otestována v prohlížečích Google Chrome 27, Internet Explorer 10 a Mozilla Firefox 21 a ve všech se chovala korektně.

Kontrola správnosti výpočtů byla provedena na cca dvaceti příkladech, které byly vyřešeny ručně, a stejné zadání bylo zpracováno vytvořenou aplikací. Aplikace automaticky spočítala správné řešení ve všech případech. V několika situacích, kdy se výsledky lišily, byly odhaleny chyby v ručním výpočtu a po přepočítání bylo dosaženo stejného výsledku, jako byl výstup aplikace.

Aplikace je v odevzdané podobě umístěna na adrese [dpp.somee.com](http://dpp.somee.com) a je možné ji zde vyzkoušet. Práce zde bude dostupná, dokud bude mít podle pravidel hostingu alespoň 5 zobrazení za měsíc. Pokud již tato stránka není dostupná, kontaktujte vedoucího práce nebo diplomanta s požadavkem na její znovuzpřístupnění.

## 6 ŘEŠENÉ PŘÍKLADY

U každého řešeného příkladu jsou pro usnadnění kontroly zobrazeny průběhy jednotlivých separovaných funkcí.

### Příklad 1

$$\text{Minimum } x_1 \log_{10}(x_1 + 1) + \frac{x_2^2}{9} + x_3 e^{x_3 - 9}$$

$$x_1 + x_2 + x_3 = 9; h = 0,5$$

bk	x1(b)	F1	x2(b)	F2	x3(b)	F3
0	0	0	0	0	-	-
0.5	0.5	0.088	0.5	0.0278	-	-
1	1	0.301	1	0.1111	-	-
1.5	1.5	0.5969	1	0.1992	-	-
2	2	0.9542	1.5	0.338	-	-
2.5	2.5	1.3602	2	0.5325	-	-
3	3	1.8062	2	0.7455	-	-
3.5	3.5	2.2862	2.5	0.9955	-	-
4	4	2.7959	2.5	1.2914	-	-
4.5	4.5	3.3316	3	1.5969	-	-
5	5	3.8908	3	1.9542	-	-
5.5	5.5	4.471	3.5	2.3154	-	-
6	6	5.0706	3.5	2.7213	-	-
6.5	6.5	5.6879	4	3.1379	-	-
7	7	6.3216	4	3.584	-	-
7.5	7.5	6.9706	4.5	4.0562	-	-
8	8	7.6339	4.5	4.5362	-	-
8.5	8.5	8.3107	4.5	5.0459	-	-
9	9	9	5	5.5737	6	1.0442

Tab. 4. Tabulka s výpočty příkladu 1

Odečtení výsledků:

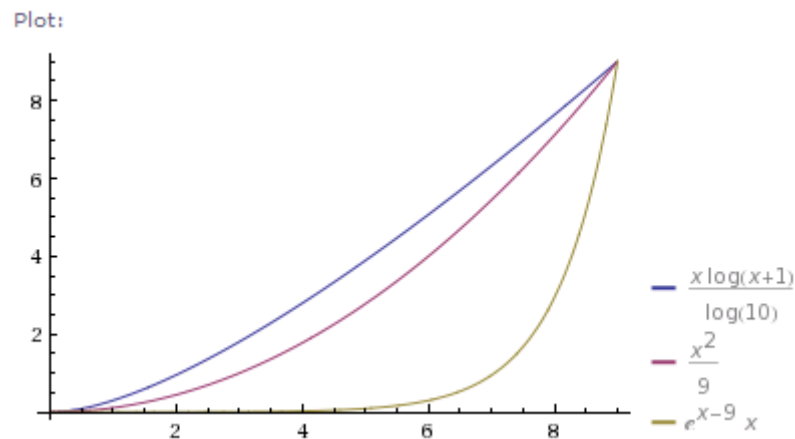
$$x_1=1$$

$$x_2=2$$

$$x_3=6$$

Hodnota funkce  $f=1,0442$

Průběh separovaných funkcí je znázorněn na obrázku 14



Obr. 14. Průběh separovaných funkcí v příkladu 1

## Příklad 2

$$\text{Maximum } \frac{1}{\cos(x_1^2)} + \sqrt{x_2} + x_3^2 \sin\left(\frac{\pi x_3}{2}\right)$$

$$x_1 + x_2 + x_3 = 1; h = 0,1$$

bk	x1(b)	F1	x2(b)	F2	x3(b)	F3
0	0	1	0	1	-	-
0.1	0.1	1.0001	0.1	1.3162	-	-
0.2	0.2	1.0008	0.2	1.4472	-	-
0.3	0.3	1.0041	0.3	1.5477	-	-
0.4	0.4	1.0129	0.4	1.6325	-	-
0.5	0.5	1.0321	0.5	1.7071	-	-
0.6	0.6	1.0685	0.6	1.7746	-	-
0.7	0.7	1.1334	0.7	1.8367	-	-
0.8	0.8	1.2467	0.8	1.8944	-	-
0.9	0.9	1.4503	0.9	1.9487	-	-
1	1	1.8508	1	2	0.9	2.1163

Tab. 5. Tabulka s výpočty příkladu 2

Odečtení výsledků:

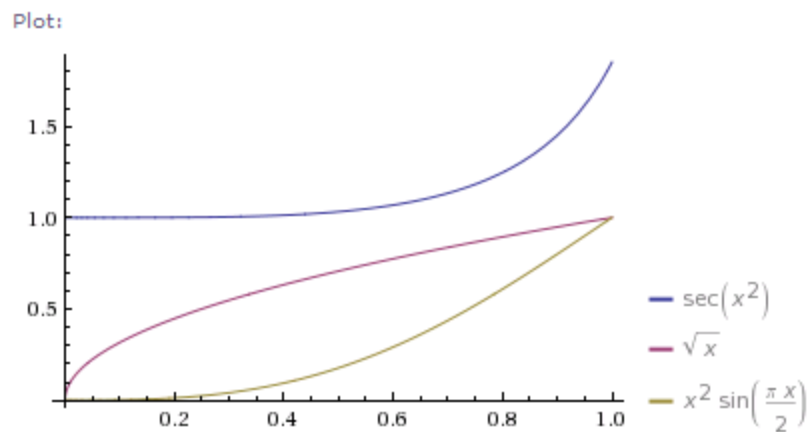
$$x_1=0$$

$$x_2=0,1$$

$$x_3=0,9$$

Hodnota funkce  $f=2,1163$

Průběh separovaných funkcí je znázorněn na obrázku 15



Obr. 15. Průběh separovaných funkcí v příkladu 2

### Příklad 3

$$\text{Minimum } x_1 + 2x_2^2 + 3x_3^3 + 4x_4^4 + 5x_5^5 + 6x_6^6$$

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 1; h = 0,1$$

bk	x1(b)	F1	x2(b)	F2	x3(b)	F3	x4(b)	F4	x5(b)	F5	x6(b)	F6
0	0	0	0	0	0	0	0	0	0	0	-	-
0.1	0.1	0.1	0.1	0.02	0.1	0.003	0.1	0.0004	0.1	0.0001	-	-
0.2	0.2	0.2	0.2	0.08	0.1	0.023	0.1	0.0034	0.1	0.0005	-	-
0.3	0.3	0.3	0.3	0.18	0.2	0.044	0.2	0.0094	0.2	0.002	-	-
0.4	0.4	0.4	0.2	0.28	0.3	0.101	0.2	0.0294	0.2	0.005	-	-
0.5	0.5	0.5	0.2	0.38	0.3	0.161	0.2	0.0504	0.2	0.011	-	-
0.6	0.6	0.6	0.3	0.48	0.3	0.261	0.3	0.0764	0.3	0.0216	-	-
0.7	0.7	0.7	0.2	0.58	0.3	0.361	0.3	0.1334	0.3	0.0416	-	-
0.8	0.8	0.8	0.2	0.68	0.3	0.461	0.3	0.1934	0.3	0.0626	-	-
0.9	0.9	0.9	0.2	0.78	0.3	0.561	0.4	0.2634	0.3	0.0886	-	-
1	1	1	0.3	0.88	0.3	0.661	0.4	0.3634	0.4	0.1276	0.3	0.0459

Tab. 6. Tabulka s výpočty příkladu 3

Odečtení výsledků:

$$x_1=0$$

$$x_2=0,1$$

$$x_3=0,1$$

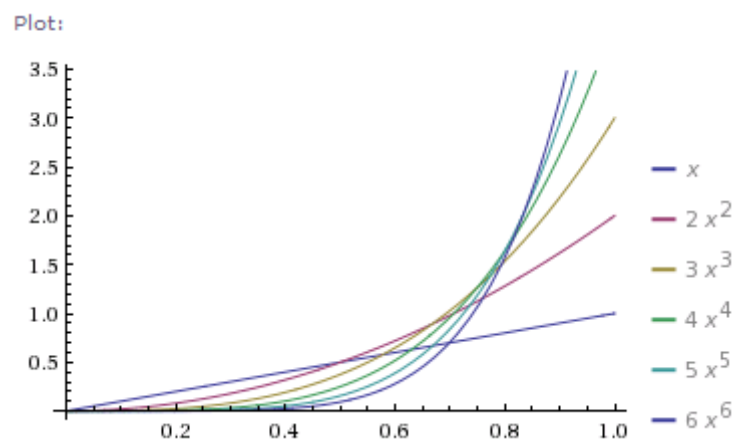
$$x_4=0,2$$

$$x_5=0,3$$

$$x_6=0,3$$

Hodnota funkce  $f=0,0459$

Průběh separovaných funkcí je znázorněn na obrázku 16



Obr. 16. Průběh separovaných funkcí v příkladu 3

#### Příklad 4

$$\text{Minimum } 40x_1^2 + x_2^2 + 10x_3^2$$

$$2x_1 + 0,2x_2 + 0,8x_3 = 8; h = 1$$

Substituce:

$$y_1=2x_1$$

$$y_2=0,2x_2$$

$$y_3=0,8x_3$$

bk	y1(b)	F1	y2(b)	F2	y3(b)	F3
0	0	0	0	0	-	-
1	1	10	0	10	-	-
2	2	40	1	35	-	-
3	3	90	1	65	-	-
4	4	160	1	115	-	-
5	5	250	1	185	-	-
6	6	360	2	260	-	-
7	7	490	2	350	-	-
8	8	640	2	460	2	322.5

Tab. 7. Tabulka s výpočty příkladu 4

Odečtení výsledků:

$$y_1=4$$

$$y_2=2$$

$$y_3=2$$

Zpětná substituce:

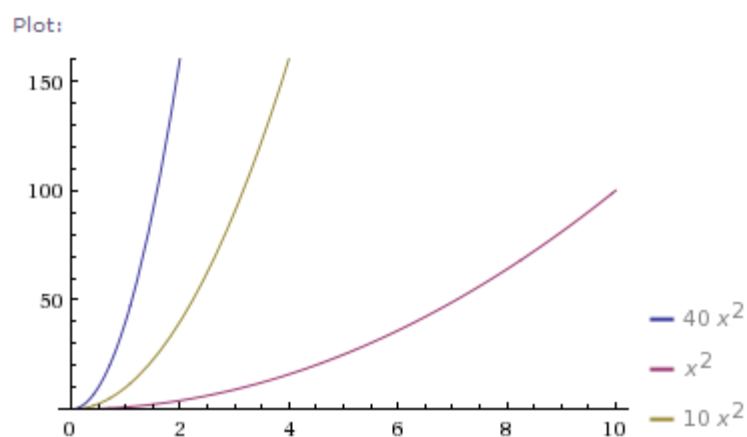
$$x_1=2$$

$$x_2=10$$

$$x_3=2,5$$

Hodnota funkce  $f=322,5$

Průběh separovaných funkcí je znázorněn na obrázku 17



Obr. 17. Průběh separovaných funkcí v příkladu 4

## Příklad 5

$$\text{Maximum } (1 - (x_1 - 0,2)^2) + (1 - (x_2 - 0,4)^2)$$

$$x_1 + x_2 \leq 1; h = 0,2$$

bk	x1(b)	F1	x2(b)	F2
0	0	0.96	0	1.8
0.2	0.2	1	0.2	1.92
0.4	0.4	0.96	0.2	1.96
0.6	0.6	0.84	0.4	2
0.8	0.8	0.64	0.4	1.96
1	1	0.36	0.6	1.92

Tab. 8. Tabulka s výpočty příkladu 5

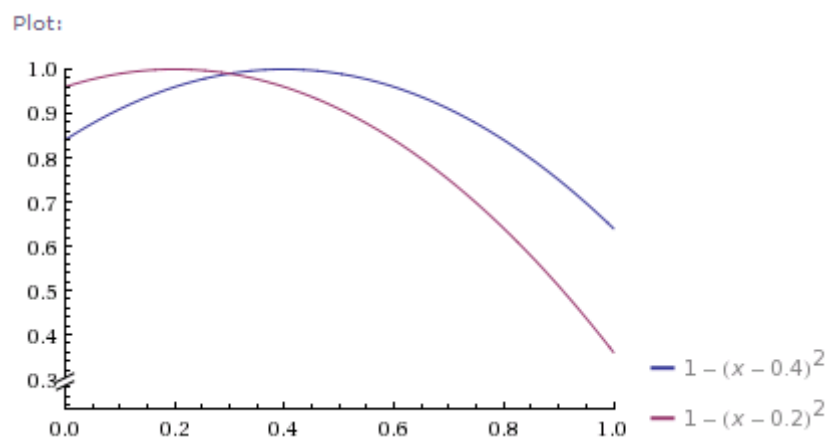
Odečtení výsledků:

$$x_1=0,2$$

$$x_2=0,4$$

Hodnota funkce  $f=2$

Průběh separovaných funkcí je znázorněn na obrázku 18



Obr. 18. Průběh separovaných funkcí v příkladu 5

## ZÁVĚR

Cílem této práce bylo podrobné vysvětlení optimalizační metody Dynamické programování, návrhnutí algoritmu řešící problém dělení zdrojů tabulkovou formou a tento algoritmus implementovat v podobě webové aplikace.

V první kapitole, věnované optimalizaci obecně, byly zopakovány základní pojmy týkající se funkcí a problémů optimalizace. Úlohy optimalizace byly rozděleny do skupin podle typu problému, které je potřeba vyřešit.

Následuje podrobný popis dynamického programování, včetně historie této metody a Bellmanova principu optimality. Jsou také vysvětleny vybrané algoritmy, které jsou založeny na metodách dynamického programování.

Třetí kapitola je věnována webovým aplikacím. Představuje úvod do této problematiky a přibližuje nejčastěji používané platformy, na kterých je možné tyto aplikace vyvíjet a provozovat. Jsou to PHP, ASP a Java. Není opomenuta ani bezpečnost webových aplikací.

Praktická část práce je rozdělena do tří kapitol. Ve čtvrté kapitole je navrhnout obecný algoritmus, pomocí kterého je možné řešit tabulkovou formou problém dělení zdrojů.

Pátá kapitola popisuje konkrétní implementaci tohoto algoritmu na platformě ASP. Vytvořená aplikace vypočítá řešení uživatelem zadané účelové funkce a omezující podmínky, umožňuje jednorázové vypsání celého postupu i s výsledky, postupné krokování řešení i s výpisem pomocných výpočtů nebo uložení výsledného řešení do PDF. Aplikace byla otestována na dvaceti příkladech, které řešily nezávislé třetí osoby, a vytvořená aplikace vždy dospěla ke správnému výsledku (u několika rozdílných výsledků byl nakonec odhalen problém v ručním řešení). Tato aplikace také odpovídá aktuálním webovým standardům, neboť výsledný HTML kód, který generuje, prošel validací W3C. Tato aplikace tedy splňuje všechny požadavky, které na ní byly v zadání kladeny. Je možné ji používat k řešení praktických úloh nebo ke kontrole ručního postupu.

Poslední kapitola je věnována sbírce řešených příkladů.

## ZÁVĚR V ANGLIČTINĚ

The aim of this thesis was to explain the optimization method Dynamic programming, design algorithm for solving the problem of Resource allocation in table form and implement this algorithm as a web application.

The first chapter is dedicated to optimization in general and repeats basic terms, concepts and problems of optimization. Optimization tasks were divided into groups according to the type of problem that must be solved.

Detailed description of the dynamic programming, including the history of the method and Bellman's principle of optimality are described in the second chapter. Selected algorithms based on the dynamic programming methods are explained.

Third chapter addresses the Web applications. It briefly describes this field and most common platforms used for development and running of these applications. These are PHP, ASP and Java. Web application security is not omitted.

Practical part of this thesis is separated into three parts. Algorithm for solving the problem of Resource allocation in table form is designed in its first part.

Fifth chapter describes an implementation of this algorithm on ASP platform. The implemented application calculates the solution of user-specified objective function and constraints. It allows user to display the final result of the calculation, gradual stepping of calculation with a display of auxiliary calculations or to save the result to PDF file. Application was tested on twenty examples solved independently by third persons. The application always calculated correct result (there were several different results, but they came out to be error in manually calculated solution). This application also meets current web standards, since the resulting HTML code passed W3C validation. This application therefore meets all the requirements required by the assignment. It can be used to solve practical problems or to check the results of manual calculation.

The last chapter is devoted to the collection of solved problems.

**SEZNAM POUŽITÉ LITERATURY**

- [1] JUREK, Miloš. *Numerické metody optimalizace*. Zlín, 2007. Diplomová práce. FAI UTB ve Zlíně.
- [2] HÁJEK, Michal. *Soubor úloh ke cvičení do předmětu Optimalizace - lineární a dynamické programování*. Zlín, 2009. Bakalářská práce. FAI UTB ve Zlíně.
- [3] Game theory. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-06-04]. Dostupné z: [http://en.wikipedia.org/wiki/Game\\_theory](http://en.wikipedia.org/wiki/Game_theory)
- [4] DREYFUS, Stuart. *Richard Bellman on the birth of Dynamic Programming*. [online]. 2002 [cit. 2013-05-27]. Dostupné z: <http://web.udl.es/usuaris/MatFDiE/OptiSim/2DPOperRes02.pdf>
- [5] BELLMAN, Richard. *The Theory of Dynamic Programming*. [online]. 1954. [cit. 2013-05-28]. Dostupné z: <http://www.rand.org/content/dam/rand/pubs/papers/2008/P550.pdf>
- [6] GUTTAG, John. *Dynamic Programming: Overlapping Subproblems, Optimal Substructure*. [přednáška]. Cambridge: MIT, 2008. [online]. [cit. 2013-06-02]. Záznam dostupný z: <http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-00-introduction-to-computer-science-and-programming-fall-2008/video-lectures/lecture-13/>
- [7] CORMEN, Thomas H., Charles E. LEISERSON, Ronald L. RIVEST a Clifford STEIN. *Introduction to algorithms*. 3. vyd. Cambridge Massachusetts: MIT Press, 2009, 1292 s. ISBN 978-0-262-03384-8.
- [8] BOGOMOLNY, Alexander. *Tower of Hanoi*. [online]. [cit. 2013-06-09]. Dostupné z: <http://www.cut-the-knot.org/recurrence/hanoi.shtml>
- [9] GAJDA, Bohumil. *Speciální metody optimalizace*. Zlín, 2009. Diplomová práce. FAI UTB ve Zlíně.
- [10] PROKOP, Roman. *Optimalizace*. [přednáška]. Zlín, 2009.
- [11] OSI model. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-06-06]. Dostupné z: [http://en.wikipedia.org/wiki/OSI\\_model](http://en.wikipedia.org/wiki/OSI_model)

- [12] Hypertext Transfer Protocol. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-06-06]. Dostupné z: [http://en.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)
- [13] *FAQ - HHttpd Wiki*. [online]. [cit. 2013-06-07]. Dostupné z: <http://wiki.apache.org/httpd/FAQ>
- [14] *March 2013 Web Server Survey*. [online]. [cit. 2013-06-07]. Dostupné z: <http://news.netcraft.com/archives/2013/03/01/march-2013-web-server-survey.html>
- [15] *Active Server Pages (ASP) support in Windows*. [online]. [cit. 2013-06-07]. Dostupné z: <http://support.microsoft.com/kb/2669020>
- [16] *Introduction to IIS Architectures*. [online]. [cit. 2013-06-07]. Dostupné z: <http://www.iis.net/learn/get-started/introduction-to-iis/introduction-to-iis-architecture>
- [17] Java (software platform). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-06-07]. Dostupné z: [http://en.wikipedia.org/wiki/Java\\_\(software\\_platform\)](http://en.wikipedia.org/wiki/Java_(software_platform))
- [18] VOJÁČEK, Petr. *SQL Injection a zabezpečení*. [online]. [cit. 2013-06-07]. Dostupné z: <http://programujte.com/clanek/2007041802-sql-injection-a-zabezpeceni/>
- [19] *Pokročilé techniky XSS*. [online]. 2008 [cit. 2013-06-08]. Dostupné z: <http://www.soom.cz/index.php?name=articles/show&aid=485&title=Pokrocile-techniky-XSS>
- [20] *ASP FPDF*. [online]. [cit. 2013-06-08]. Dostupné z: <https://sites.google.com/site/asfpdf/>

**SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK**

AJAX	Asynchronous Javascript And XML
ASP	Active Server Pages.
CLI	Common Language Infrastructure.
HTML	Hyper Text Markup Language.
HTTP	Hyper Text Transfere Protocol.
IIS	Internet Information Services.
OOP	Objektově Orientované Programování
PHP	PHP: Hypertext Preprocessor
RAND	Research ANd Development.
URL	Uniform Resource Locator
WWW	World Wide Web.
XSS	Cross-site Scripting

**SEZNAM OBRÁZKŮ**

Obr. 1. Ilustrace určení nejkratší trasy pomocí Dijkstrova algoritmu .....	18
Obr. 2. Výchozí stav .....	20
Obr. 3. Uvolnění spodního kotouče přesunutím všech ostatních .....	20
Obr. 4. Poté přesunout největší kotouč .....	20
Obr. 5. A nakonec zbývající kotouče přesunout na cílový kolík .....	20
Obr. 6. Schéma fungování aplikace .....	33
Obr. 7. Postup výpočtu .....	34
Obr. 8. Vývojový diagram aplikace .....	37
Obr. 9. Úvodní strana aplikace .....	40
Obr. 10. Jednorázový výpis řešení .....	42
Obr. 11. Ukázka PDF orientovaného na výšku .....	42
Obr. 12. Ukázka PDF orientovaného na šířku .....	43
Obr. 13. Postupný výpis řešení s pomocnými výpočty .....	44
Obr. 14. Průběh separovaných funkcí v příkladu 1 .....	46
Obr. 15. Průběh separovaných funkcí v příkladu 2 .....	47
Obr. 16. Průběh separovaných funkcí v příkladu 3 .....	48
Obr. 17. Průběh separovaných funkcí v příkladu 4 .....	49
Obr. 18. Průběh separovaných funkcí v příkladu 5 .....	50

**SEZNAM TABULEK**

Tab. 1. Tabulka s výpočty příkladu dělení zdrojů .....	22
Tab. 2. Tabulka s výpočty příkladu dělení zdrojů se substitucí.....	24
Tab. 3. Struktura souborů v implementované aplikaci .....	36
Tab. 4. Tabulka s výpočty příkladu 1 .....	45
Tab. 5. Tabulka s výpočty příkladu 2 .....	46
Tab. 6. Tabulka s výpočty příkladu 3 .....	47
Tab. 7. Tabulka s výpočty příkladu 4 .....	49
Tab. 8. Tabulka s výpočty příkladu 5 .....	50

## SEZNAM PŘÍLOH

P I DVD se zdrojovými kódy