VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DIFF PRO RŮZNÉ TYPY DOKUMENTŮ

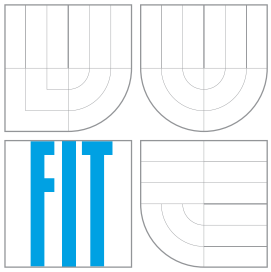
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL ZEMKO

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DIFF PRO RŮZNÉ TYPY DOKUMENTŮ

MULTIPLE DOCUMENT TYPE DIFF

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL ZEMKO
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR CHMELAŘ
SUPERVISOR

BRNO 2009

Abstrakt

Při práci na důležitých projektech často pravidelně zálohujeme průběžné verze tohoto projektu. Někdy však potřebujeme zjistit, jaké změny jsme provedli mezi dvěma verzemi. Pro zjištení těchto změn nám slouží program diff, který porovná dva textové soubory po řádcích a umí zobrazit rozdíly v různych formátech. Problém nastává pokud chceme porovnat jiné než textové soubory. Cílem této práce je vytvořit jeden nástroj, který dovede porovnávat různé typy souborů, například obrázky, soubory kancelářskeho balíku Open Office, zdrojové texty DTP systému LaTeX, konfigurační soubory a samozřejmě i prosté textové soubory. Každý z uvedených typů souborů je něčím specifický a vyžaduje si individální přístup. Táto práce se zabývá teoretickým rozborem problému a následující implementaci samotného programu, který realizuje porovnávání. Závěr obsahuje zhodncení vytvořené aplikace z hlediska rychlosti. Program bude nasazen v repozitářích RedHat.

Abstract

During the work on important projects, we have to backup current versions periodically. But sometimes we want to know what changes between two versions has been made. To recognize these changes, we can use the diff program that compares two text files line by line and can show differencies in various formats. A problem occurs, if we want to compare other than text files. The purpose of this bachelor thesis is to create one tool that can compare different types of files, for example images, Open Office files, DTP system LaTeXfiles, config files and, of course, standard text files. Each of listed types is specific and demands unique approach. This thesis deals with theoretical analysis of the problem and also with realization of the program which executes the comparision.

Klíčová slova

diff, porovnávaní, text, open office, konfigurační soubor, latex, obrázek, LCS

Keywords

diff, compare, text, open office, config file, latex, image, LCS

Citace

Michal Zemko: Diff pro různé typy dokumentů, bakalářská práce, Brno, FIT VUT v Brně, 2009

Diff pro různé typy dokumentů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Chmelaře

.....
Michal Zemko
19. května 2009

Poděkování

Děkuji vedoucímu práce Ing. Petrovi Chmelařovi za odbornou pomoc a pedagogické vedení při řešení mé bakalářské práce.

Obsah

Kapitola 1

Úvod

V sedemdesiatych rokoch, pri rozvoji informatiky, vznikla požiadavka na nástroj, ktorý by dokázal porovnať dva súbory a zobraziť rozdiely medzi nimi. Dôvodov bolo viac, napríklad pri vývoji programov sa často pravidelne zálohovali priebežné verzie zdrojových kódov a niekedy bolo potrebné zistiť, aké zmeny medzi dvoma verziami boli prevedené. Alebo pri šírení nových verzií súborov stačilo posielať iba zmeny voči predchádzajúcej verzii. Na túto požiadavku reagoval Douglas McIlroy, ktorý priviedol na svet program `diff`. Tento program porovnával dva textové súbory riadok po riadku a zobrazil ich rozdiel. Takto mohli programátori kontrolovať zmeny medzi jednotlivými verziami zdrojových kódov, respektíve šíriť nové verzie pomocou rozdielov voči predchádzajucej. Neskôr bol `diff` zaradený do systému UNIX, postupne zdokonaľovaný a používa sa s obľubou dodnes.

Problém nastáva, ak chceme porovnať iné ako textové súbory. Cieľom tejto práce je vytvoriť jeden nástroj, ktorý dokáže porovnávať rôzne druhy súborov, napríklad obrázky, súbory kancelárskeho balíku Open Office, zdrojové texty DTP systému LaTeX, konfiguračné súbory a samozrejme aj prosté textové súbory. Každý z uvedených typov súborov je niečim špecifický a vyžaduje si osobitý prístup. Táto práca sa zaoberá teoretickým rozborom problému a následnou realizáciou samotného programu, ktorý bude porovnávanie realizovať. Téma práce vznikla v spolupráci FIT VUT v Brně a spoločnosti RedHat.

Srdcom programu na porovnávanie súborov je algoritmus, ktorý nájde spoločné časti súborov. Pretože ak nájdeme časti, ktoré sú spoločné, dokážeme zobraziť rozdiely. Tento algoritmus, ktorý nájde najdlhšiu spoločnú subsekvenciu si popíšeme v druhej kapitole. Spolu s ním si uvedieme aj algoritmy na porovnávanie reťazcov.

V tretej kapitole sa zameriame na súčasné programy na porovnávanie, najmä na balík GNU diffutils, ktorý je veľmi používaný a je východiskový pre náš program. Program `diff` je konzolová aplikácia, preto si predvedieme aj program s grafickým užívateľským rozhraním.

Pretože budeme vytvárať program, ktorý pracuje s rôznymi formátmi súborov, popíšeme si v štvrtej kapitole jednotlivé typy dokumentov, ktoré budeme porovnávať. Hlavne ich štruktúru a charakteristické znaky.

Ďalšia kapitola sa zaoberá analýzou a návrhom samotnej aplikácie. V analýze si rozdelíme celý problém na čiastkové podproblémy a pomocou poznatkov z predchádzajúcich kapitol navrhneme možné riešenie.

V predposlednej kapitole stručne opíšeme implementáciu programu a jeho jednotlivých modulov. Dostatočnú pozornosť budeme venovať testovaniu vytvoreného prototypu programu.

Nakoniec si v závere zhrnieme dosiahnuté výsledky a uvedieme možné budúce rozšírenia programu.

Kapitola 2

Algoritmy

V programoch na porovnávanie textových súborov sa využívajú dva základné algoritmy:

- Algoritmus, ktorý nájde najdlhší spoločný podreťazec oboch súborov. 2.1
- Algoritmus, ktorý určí, v akej miere sú dva reťazce podobné. 2.2

V tejto kapitole si popíšeme základný algoritmus na nájdenie najdlhšieho podreťazca, ktorý sa nachádza v oboch reťazcoch. V anglickej literatúre sa môžeme stretnúť s týmto algoritmom pod názvom „Longest common subsequence“ (LCS). Tento algoritmus je klasický informatický problém a je základom pre program `diff` a jemu podobné. Taktiež má veľké uplatnenie v bioinformatike.[10]

Algoritmus LCS je dôležitý pri porovnávaní dvoch textov a hľadaní rozdielov medzi nimi. Ak nájdeme časti súborov, ktoré sú spoločné pre oba súbory, dokážeme veľmi ľahko získať zoznam zmien medzi oboma súbormi. Tento problém si vysvetlíme na nasledujúcom príklade:

Máme dve sekvencie znakov: `a b c d f g h j q z` a `a b c d e f g i j k r x y z`. Chceme nájsť najdlhšiu možnú postupnosť znakov, obsiahnutých v oboch sekvenciách. Táto spoločná sekvencia znakov musí mať rovnaké poradie znakov ako v oboch sekvenciách. V našom prípade je spoločná sekvencia `a b c d f g j z`. Ak porovnáme túto sekvenciu s prvou sekvenciou, zistíme, ktoré znaky sa nenachádzajú v druhej sekvencii, čiže boli zmazané. Keď porovnáme sekvenciu spoločných znakov s druhou sekvenciou, zistíme, ktoré znaky nie sú obsiahnuté v prvej sekvencii, čiže boli do druhej sekvencie pridané. Na tomto princípe pracujú programy na porovnávanie súborov. Po porovnaní spoločnej sekvencie s oboma pôvodnými sekvenciami znakov získame rozdiel oboch súborov. Ak označíme znaky, ktoré sú jedinečné len v prvom súbore znakom - a znaky jedinečné v druhom súbore znakom +, dostaneme následovný výstup: [5]

```
e   h i   q   k r x y
+   - +   -   + + + +
```

Ak však pre nás nie je podstatné, ktoré časti reťazcov sú odlišné a zaujíma nás iba miera odlišnosti vyjadrená číslom, dokážeme túto mieru odlišnosti vyjadriť pomocou Hammingovej vzdialenosti, respektíve pomocou Levenshteinovej vzdialenosti.

V následujúcich dvoch podkapitolách si vysvetlíme jednotlivé algoritmy. Zameriame sa predovšetkým na algoritmus nájdenia najdlhšieho spoločného podreťazca, ktorý je z hľadiska porovnávania súborov najdôležitejší.

2.1 Najdlhší spoločný podreťazec

V tejto kapitole bude algoritmus LCS popísaný podľa pána Cormena [2].

V bioinformatických aplikáciach potrebujeme často porovnávať DNA dvoch (alebo viac) odlišných organizmov. Vzor DNA obsahuje reťazec molekúl nazývaných bázy. Možné bázy sú adenín, guanín, cytosín a thymín. Reprezentujúc každú z týchto báz ich počiatočným písmenom, môže byť vzor DNA vyjadrený ako reťazec nad konečnou množinou $\{A, C, G, T\}$. Napríklad, DNA jedného organizmu môže byť $S_1 = ACCGGTCGAGTGCGCGGAAGCC GGCCGAA$, zatiaľ čo DNA iného organizmu môže byť $S_2 = GTCGTTCGGAATGCCGTT GCTCTGTAAA$. Jedným z cieľov porovnávania dvoch vzorov DNA je určiť, ako „podobné“ si dva vzory DNA sú, ako určité meradlo, ako veľmi príbuzné tie dva organizmy sú. Podobnosť môže byť a je definovaná rôznymi spôsobmi. Napríklad, môžeme povedať, že dva vzory DNA sú si podobné, ak jeden je podreťazec druhého. V našom prípade, ani jeden z S_1 a S_2 nie je podreťazcom druhého. Alternatívne by sme mohli povedať, že dva vzory DNA sú si podobné, ak počet zmien potrebných k transformácii jedného do druhého je malý. Avšak iný spôsob merania podobnosti vzorov S_1 a S_2 je nájdenie tretieho vzoru S_3, ktorého bázy sa objavujú v každom zo vzorov S_1 a S_2. Tieto bázy sa musia vyskytovať v rovnakom poradí, ale nie nezbytne postupne. Dlhší vzor S_3 môžeme nájsť, ak sú si S_1 a S_2 viac podobné. V našom prípade, najdlhší vzor S_3 je $GTCGTCGGAAGCCGGCCGAA$.

Formalizujme problém nájdenia najdlhšieho spoločného podreťazca. Podreťazec daného reťazca je iba daný reťazec s nula alebo viac vynechanými elementami. Formálne, je daná sekvencia $X = \langle x_1, x_2, \ldots, x_m \rangle$, iná sekvencia $Z = \langle z_1, z_2, \ldots, z_k \rangle$ je **subsekvencia** X, ak existuje stúpajúca sekvencia $\langle i_1, i_2, \ldots, i_k \rangle$ indexov X taká, že pre všetky $j = 1, 2, \ldots, k$ platí $x_{i_j} = z_j$. Napríklad, $Z = \langle B, C, D, B \rangle$ je subsekvencia sekvencie $X = \langle A, B, C, B, D, A, B \rangle$ s korešpondujúcou postupnosťou indexov $\langle 2, 3, 5, 7 \rangle$.

Pre dané dve sekvencie X a Y môžeme povedať, že sekvencia Z je **spoločná subsekvencia** sekvencií X a Y, ak je Z podsekvencia X aj Y. Napríklad ak $X = \langle A, B, C, B, D, A, B \rangle$ a $Y = \langle B, D, C, A, B, A \rangle$, sekvencia $\langle B, C, A \rangle$ je spoločná podsekvencia oboch sekvencií X aj Y. Sekvencia $\langle B, C, A \rangle$ nie je najdlhšia spoločná podsekvencia sekvencií X a Y, pretože má dĺžku 3 a sekvencia $\langle B, C, B, A \rangle$, ktorá je spoločná v oboch sekvenciách X aj Y má dĺžku 4. Sekvencia $\langle B, C, B, A \rangle$ je najdlhšia spoločná subsekvencia sekvencií X a Y, takisto ako aj sekvencia $\langle B, D, B, A \rangle$. Ďalej už neexistuje spoločná subsekvencia dĺžky 5 alebo dlhšej.

Pri hľadaní **najdlhšej spoločnej subsekvencie** máme dané dve sekvencie $X = \langle x_1, x_2, \ldots, x_m \rangle$ a $Y = \langle y_1, y_2, \ldots, y_n \rangle$ a prajeme si nájsť, čo najdlhšiu spoločnú subsekvenciu sekvencií X a Y. V tejto časti dokážeme, že problém nájdenia najdlhšej spoločnej subsekvencie môže byť vyriešený efektívne použitím dynamického programovania.

2.1.1 Charakteristika najdlhšej spoločnej subsekvencie

Prístup k riešeniu LCS problému hrubou silou znamená vytvoriť všetky subsekvencie sekvencie X a skontrolovať každú takúto subsekvenciu, či nie je tiež subsekvencia sekvencie Y. Každá subsekvencia sekvencie X odpovedá podmnožine indexov $\{1, 2, \ldots, m\}$ sekvencie X. Takto dostaneme 2^m subsekvencií sekvencie X, takže takýto prístup vyžaduje exponenciálny čas $\Theta(n2^m)$, ktorý je nepraktický pre dlhé sekvencie.

LCS problém má „optimálnu štruktúru“, ktorú dokazuje následujúca veta 2.1.1. Optimálna štruktúra znamená, že problém môžeme rozdeliť na menšie jednoduchšie podproblémy, a tak ďalej, až nakoniec sa riešenie stane triviálne. Ako môžeme vidieť, prirodzené triedy podproblémov zodpovedajú párom „prefixov“ dvoch vstupných sekvencií. Ak máme

byť dôsledný, je daná sekvencia $X = \langle x_1, x_2, \ldots, x_m \rangle$, definujeme i-tý **prefix** sekvencie X, pre $i = 0, 1, \ldots, m$ ako $X_i = \langle x_1, x_2, \ldots, x_i \rangle$. Napríklad, ak $X = \langle A, B, C, B, D, A, B \rangle$, potom $X_4 = \langle A, B, C, B \rangle$ a X_0 je prázdna sekvencia.

Veta 2.1.1 *Optimálna štruktúra LCS*

Nech $X = \langle x_1, x_2, \ldots, x_m \rangle$ a $Y = \langle y_1, y_2, \ldots, y_n \rangle$ sú sekvencie, a nech $Z = \langle z_1, z_2, \ldots, z_k \rangle$ je nejaká LCS sekvencií X a Y.

1. Ak $x_m = y_n$, potom $z_k = x_m = y_n$ a Z_{k-1} je LCS sekvencií X_{m-1} a Y_{n-1}.
2. Ak $x_m \neq y_n$, potom $z_k \neq x_m$ značí, že Z je LCS sekvencie X_{m-1} a Y.
3. Ak $x_m \neq y_n$, potom $z_k \neq y_n$ značí, že Z je LCS sekvencie X a Y_{n-1}.

Formulácia vety 2.1.1 ukazuje, že LCS dvoch sekvencií obsahuje v sebe LCS prefixov dvoch sekvencií. Takže ako sme mohli vidieť, LCS problém má optimálnu štruktúru.

2.1.2 Rekurzívne riešenie

Veta 2.1.1 naznačuje, že keď hľadáme LCS sekvencií $X = \langle x_1, x_2, \ldots, x_m \rangle$ a $Y = \langle y_1, y_2, \ldots, y_n \rangle$, overíme buď jeden, alebo dva podproblémy. Ak $x_m = y_n$, musíme nájsť LCS sekvencií X_{m-1} a Y_{n-1}. Pripojením $x_m = y_n$ do tejto LCS získame LCS sekvencií X a Y. Ak $x_m \neq y_n$, potom musíme riešiť dva podproblémy: hľadať LCS sekvencií X_{m-1} a Y a LCS sekvencií X a Y_{n-1}. Tá sekvencia, ktorá bude z týchto dvoch sekvencií dlhšia, bude LCS sekvencií X a Y. Pretože tieto prípady vyčerpali všetky možnosti, vieme, že jeden z optimálnych podproblémov riešenia musí byť použitý v LCS sekvencií X a Y.

Môžeme ľahko vidieť prekrývajúce sa podproblémy pri hľadaní LCS. K nájdeniu LCS sekvencií X a Y možno budeme potrebovať nájsť LCS sekvencií X a Y_{n-1} a sekvencií X_{m-1} a Y. Ale každý z týchto podproblémov obsahuje podproblém nájsť LCS sekvencií X_{m-1} a Y_{n-1}.

Tak ako pri maticovom násobení, naše rekurzívne riešenie LCS problému zahrňuje stanovenie opakovania pre hodnotu optimálneho riešenia. Definujme $c[i, j]$ ako dĺžku LCS sekvencií X_i a Y_j. Ak $i = 0$ alebo $j = 0$, jedna zo sekvencií má nulovú dĺžku, takže LCS má dĺžku 0. Optimálnu subštruktúru LCS problému vracia následujúci rekurzívny vzorec.

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max(c[i, j-1], (c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j, \end{cases} \tag{2.1}$$

Všimnime si, že v tejto rekurzívnej formulácii, podmienka v probléme obmedzuje, nad ktorými podproblémami môžeme brať ohľad. Keď $x_i = y_j$ môžeme a mali by sme uvažovať nad podproblémom hľadania LCS sekvencií X_{i-1} a Y_{j-1}. Inak namiesto toho uvažujeme dva podproblémy hľadania LCS sekvencií X_i a Y_{j-1} a sekvencií X_{i-1} a Y_j. V predchádzajúcom dynamicky programovanom algoritme sme skúmali, že žiaden podproblém nebol vylúčený kvôli podmienkam v probléme. Hľadanie LCS nie je iba dynamicky programovaný algoritmus, ktorý vylúčil podproblémy založené na podmienkach problému.

```
LCS–Length(X,Y)
    m ← len[X]
    n ← len[Y]
    for i ← 1 to m
        do c[i,0] ← 0
    for j ← 0 to n
        do c[0,j] ← 0
    for i ← 1 to m
        do for j ← 1 to n
            do if x_i = y_i
                    then c[i,j] ← c[i−1,j−1] + 1
                         b[i,j] ← "↖"
                    else if c[i−1,j] ≥ c[i,j−1]
                            then c[i,j] ← c[i−1,j]
                                 b[i,j] ← "↑"
                            else c[i,j] ← c[i,j−1]
                                 b[i,j] ← "←"
    return b and c
```

Algoritmus 2.1: Algoritmus na zistenie LCS

2.1.3 Výpočet dĺžky najdlhšej spoločnej subsekvencie

Na základe rovnice 2.1 by sme mohli ľahko napísať rekurzívny algoritmus pracujúci v exponenciálnom čase, ktorý vypočíta dĺžku LCS dvoch sekvencií. Avšak na vyriešenie tohoto problému môžeme použiť dynamické programovanie a výrazne znížiť zložitosť na $\Theta(mn)$.

Funkcia `LCS-length` prijíma ako argumenty dve sekvencie: $X = \langle x_1, x_2, \ldots, x_m \rangle$ a $Y = \langle y_1, y_2, \ldots, y_n \rangle$. Funkcia ukladá hodnoty $c[i, j]$ do tabuľky $c[0..m, 0..n]$, ktorej položky sú počítané po riadkoch. Funkcia udržuje tiež tabuľku $b[1..m, 1..n]$, kvôli jednoduchému zostrojeniu optimálneho riešenia. Intuitívne, $b[i, j]$ ukazuje na položku tabuľky zodpovedajúcu optimálnemu riešeniu podproblému vybraného počas počítania $c[i, j]$. Funkcia vracia tabuľky b a c. Položka $c[m, n]$ obsahuje dĺžku LCS sekvencií X a Y. Obrázok 2.1 zobrazuje tabuľky b a c, vypočítané pomocou algoritmu LCS, na sekvenciách $X = \langle A, B, C, B, D, A, B \rangle$ a $Y = \langle B, D, C, A, B, A \rangle$.

Čas behu funkcie na výpočet LCS je $\Theta(mn)$, pre každý údaj v tabuľke sa počíta so zložitosťou $\Theta(1)$. Políčko v riadku i a v stĺpci j obsahuje hodnotu $c[i, j]$ a príslušnú šipku hodnoty $b[i, j]$. Údaj 4 v políčku $c[7, 6]$ – pravý spodný roh tabuľky – je dĺžka LCS $\langle B, C, B, A \rangle$ sekvencií X a Y. Pre každé $i, j > 0$, údaj $c[i, j]$ závisí iba na $x_i = y_j$ a hodnotách v políčkach $c[i - 1, j]$, $c[i, j - 1]$ a $c[i - 1, j - 1]$, ktoré sú vypočítané pred $c[i, j]$. Na rekonštrukciu elementov LCS nasledujeme šipky v $b[i, j]$ od pravého dolného rohu. Cesta je na obrázku zvýraznená. Každá šipka $\nwarrow$ na ceste zodpovedá údaju (zvýraznenému), pre ktorý platí $x_i = y_j$, ktorý je členom LCS.

2.1.4 Vytvorenie najdlhšej spoločnej subsekvencie

Tabuľka b vrátená algoritmom hľadajúcim LCS môže byť použitá pre rýchle zostavenie LCS sekvencií $X = \langle x_1, x_2, \ldots, x_m \rangle$ a $Y = \langle y_1, y_2, \ldots, y_n \rangle$. My jednoducho začneme na políčku $b[m, n]$ a prechádzame cez tabuľku následujúc šipky. Vždy, keď narazíme na $\nwarrow$ v políčku

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	←1	↖ 1
2	B	0	↖ 1	←1	←1	↑ 1	↖ 2	←2
3	C	0	↑ 1	↑ 1	↖ 2	←2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	←3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Obrázek 2.1: Tabuľky b a c vypočítané pomocou algoritmu LCS

$b[i, j]$, vieme, že $x_i = y_j$ a tak získame ďalší element LCS. Elementy LCS získané touto metódou sú v obrátenom poradí.

2.2 Porovnávanie reťazcov

Ak nepotrebujeme vedieť aké zmeny nastali medzi dvoma reťazcami, ale stačí nám iba číslo vyjadrujúce ako veľmi sú dva reťazce podobné, môžeme použiť jeden z nasledujúcich algoritmov.

2.2.1 Hammingova vzdialenosť

Hammingova vzdialenosť medzi dvoma reťazcami je rovná počtu zmenených znakov na korešpondujúcich pozíciách v rovnako dlhých reťazcoch. Toto číslo vyjadruje minimálny počet zmien, pomocou ktorých dostaneme z prvého reťazca druhý. V inom význame je toto číslo rovné počtu chýb, ktoré transformovali prvý reťazec do druhého. Hammingovu vzdialenosť si ukážeme na príklade. Máme reťazce `akre538sk4558sl` a `akrq53rsk4118sl`. Pre tieto reťazce je Hammingova vzdialenosť 4. Presnú definíciu Hammingovej vzdialenosti nájdeme v [3].

2.2.2 Levenshteinova vzdialenosť

Levenshteinova vzdialenosť, tiež známa pod názvom „transformačná vzdialenosť“, medzi dvoma reťazcami je daná minimálnym počtom zmien, potrebných k transformácii jedného reťazca na druhý. Transformácie znamenajú vloženie, zmazanie alebo zámenu jedného

znaku. Levenshteinovu vzdialenosť si popíšeme podľa [6].

Napríklad Levenshteinova vzdialenosť medzi slovami „kitten“ a „sitting“ je 3, pretože následujúce tri zmeny prvého slova vytvoria druhé slovo. Neexistuje iný spôsob s menej než tromi zmenami.

1. kitten ← sitten (zámena „k“ za „s“)
2. sitten ← sittin (zámena „e“ za „i“)
3. sittin ← sitting (vloženie „g“ na koniec slova)

Levenshteinova vzdialenosť sa považuje za zobecnenú Hammingovu vzdialenosť, ktorá sa používa pre reťazce rovnakej dĺžky a používa iba zámenu znakov.

Algoritmus

Bežne používaný dynamicky programovaný algoritmus na vypočítanie Levenshteinovej vzdialenosti vyžaduje použitie matice o veľkosti $(n+1)\times(m+1)$, kde m a n sú dĺžky dvoch reťazcov. Tento algoritmus je založený na Wagner-Fischerovom algoritme. Nasledujúca funkcia `LevenshteinDistance` 2.2 berie dva reťazce, s dĺžky m a t dĺžky n a vypočíta Levenshteinovu vzdialenosť medzi nimi.

```
int LevenshteinDistance(char s[], char t[])
    // d je tabuľka s ~m+1 riadkami a n+1 stĺpcami
    int d[m][n];

    for (i = 0; i < m; i++)
        d[i][0] = i;
    for (j= 0; j < n; j++)
        d[0][j] = j;
    for (i = 1; i < m; i++)
        for (j= 1; j < n; j++)
            {
             if (s[i] == t[j])
                 cost = 0;
             else
                 cost = 1;

             d[i][j] = minimum(
                                d[i-1][j] + 1, // vloženie
                                d[i][j-1] + 1, // zmazanie
                                d[i-1][j-1] + cost // zámena
                               );
            }
    return d[m][n]
```

Algoritmus 2.2: Funkcia na výpočet Levenshteinovej vzdialenosti dvoch reťazcov.

Pri použití funkcie na slová uvedené v príklade, dostaneme výpis uvedený v tabuľke 2.1. Kroky, ktoré vedú k transformácii reťazca sú podtrhnuté. V políčku v pravom spodnom rohu je číslo, zodpovedajúce Levenshteinovej vzdialenosti daných reťazcov.

	0	*1*	*2*	*3*	*4*	*5*	*6*
0		k	i	t	t	e	n
1	s	1	2	3	4	5	6
2	i	2	1	2	3	4	5
3	t	3	2	1	2	3	4
4	t	4	3	2	1	2	3
5	i	5	4	3	2	2	3
6	n	6	5	4	3	3	2
7	g	7	6	5	4	4	3

Tabulka 2.1: Výstup funkcie na výpočet Levenshteinovej vzdialenosti

Kapitola 3

Programy na porovnávanie súborov

3.1 GNU Diffutils

Medzi najznámejšie a najstaršie programy na porovnávanie súborov patrí skupina programov GNU Diffutils. V čase písania tejto práce je k dispozícii verzia 2.8.1 z 5. apríla 2002. GNU Diffutils pozostáva zo štyroch samostatných programov:

- diff – nájde rozdiely medzi dvoma súbormi
- diff3 – trojcestné porovnanie súborov riadok po riadku
- sdiff – porovná dva súbory a zobrazí ich vedľa seba
- cmp – porovná dva súbory bajt po bajte

3.1.1 Popis

Je viac spôsobov ako chápať rozdiely medzi súbormi. Jeden zo spôsobov ako pochopiť rozdiely medzi súbormi je zoznam riadkov, ktoré boli zmazané, vložené alebo zmenené. Po aplikácii týchto zmien dostaneme druhý súbor. `GNU diff` porovnáva dva súbory riadok po riadku, hľadá skupiny riadkov, ktoré sú odlišné a vypíše každú takúto skupinu. Dokáže vypisovať rozdielne riadky v rôznych formátoch, ktoré majú rôzne účely. `GNU diff` vie zobraziť, či sú súbory rozdielne bez prihliadnutia na niektoré odlišnosti ako napríklad veľkosť písmen alebo počet bielych znakov (medzery, tabulátory, prázdne riadky).

Iný spôsob ako chápať rozdiely medzi súbormi je zoznam párov bajtov, ktoré môžu byť rovnaké alebo odlišné. Program `cmp` vypíše rozdiely medzi dvoma súbormi bajt po bajte namiesto riadok po riadku. Z toho vyplýva, že `cmp` je vhodnejší než `GNU diff` pri porovnávaní binárnych súborov. Pre textové súbory je `cmp` užitočný hlavne vtedy, ak chceme vedieť, či sú dva súbory identické alebo či je jeden prefixom druhého.

Program `diff3` bežne porovnáva tri vstupné súbory riadok po riadku, hľadá skupiny riadkov, ktoré sú rôzne a vypíše ich. Jeho výstup je navrhnutý tak, aby bolo ľahké preskúmať dve rôzne verzie toho istého súboru.

Pre názornejšie zobrazenie rozdielu dvoch dokumentov môžeme použiť program `sdiff`, ktorý nám zvisle rozdelí obrazovku na dve časti. V ľavej časti zobrazí prvý dokument, v pravej časti zobrazí druhý dokument a v deliacom priestore medzi nimi zobrazí znaky,

ktoré indikujú zmeny spôsobené zmazaním, pridaním alebo zmenením riadku. Takto máme prehľadne zobrazené oba texty jeden vedľa druhého.

3.1.2 História

Program `diff` bol vytvorený na počiatku sedemdesiatych rokov minulého storočia na operačnom systéme UNIX, ktorý sa objavoval v AT&T Bell Labs v Murray Hill, New Jersey. Finálnu verziu, zahrnutú do 5. vydania Unixu v roku 1974, napísal Douglas McIlroy. Tento výskum publikoval v roku 1976 v spolupráci s Jamesom W. Huntom, ktorý vyvíjal počiatočný prototyp programu `diff` [4]. McIlroyova práca predchádzala a bola ovplyvňovaná porovnávacím programom Steva Johnsona na operačnom systéme GECOS a programom `proof` Mika Leska. `proof` tiež vznikol na UNIXe, podobne ako program `diff`, produkoval riadkové zmeny a dokonca používal ostré zátvorky > a < pre vyjadrenie vloženia a zmazania riadku na výstupe programu. Heuristika použitá v týchto prvých aplikáciach bola považovaná za nespoľahlivú. Potenciálny úspech programu `diff` povzbudil McIlroya do výskumu a návrhu robustnejšieho nástroja, ktorý môže byť použitý v rôznych úlohách, ale splní výkonové limity minipočítača PDP-11. Jeho prístup k problému vyústil k spolupráci jednotlivcov z Bell Labs ako Alfred Aho, Elliot Pinson, Jeffrey Ullman a Harold S. Stone.

Dnešnú verziu programu `GNU diff` napísali Paul Eggert, Mike Haertel, David Hayes, Richard Stallman a Len Tower. Wayne Davison navrhol a implementoval unifikovaný výstupný formát. Základný algoritmus, ktorý nástroje GNU Diffutils používajú, je popísaný v článku Eugena Myersa [17] a v článku Millera Webba a Eugene Myersa [16]. Algoritmus bol tiež nezávisle objavený a popísaný v článku E. Ukkonena [19].

3.1.3 Formáty výstupu

`GNU diff` má niekoľko navzájom vylučujúcich sa volieb pre formát výstupu. V nasledujúcom texte popíšeme každý z formátov. V prílohe A si názorne tieto formáty predvedieme.

Normálny formát výstupu

Normálny formát výstupu programu `GNU diff` vypisuje každú skupinu rozdielnych riadkov bez obklopujúceho kontextu. Niekedy je takýto výstup najlepší spôsob ako ukázať, ktoré riadky boli zmenené bez rozptýlenia okolitými nezmenenými riadkami (avšak takýto výsledok môžeme dosiahnúť aj pri kontextovom alebo unifikovanom formáte, pri použití 0 riadkov ako kontextu). Normálny formát výstupu je implicitný kvôli kompatibilite so staršími verziami `GNU diffu` a štandardu POSIX.

Normálny formát výstupu pozostáva z jedného alebo viac skupín odlišných riadkov. Každá skupina predstavuje jednu oblasť, kde sú súbory rozdielne. Normálny formát výstupu vyzerá následovne:

```
značka zmeny
< súbor1-riadok
< súbor1-riadok...
---
> súbor2-riadok
> súbor2-riadok...
```

Existujú tri typy značiek zmeny. Každá obsahuje číslo riadku alebo čiarkou oddelený rozsah riadkov v prvom súbore, jeden znak indikujúci typ zmeny, ktorá nastala a číslo riadku alebo čiarkou oddelený rozsah riadkov v druhom súbore. Typy značiek zmien:

- `lar` – pridané riadky v rozsahu `r` z druhého súboru za riadok `l` z prvého súboru.
- `fct` – vymenené riadky prvého súboru rozsahu `f` za riadky druhého súboru rozsahu `t`.
- `ldr` – zmazané riadky v rozsahu `r` z prvého súboru za riadkom `l` v druhom súbore.

Každý riadok začína znakom `>` alebo `<` nasleduje medzera a text daného riadku v originálnom súbore. Znaky `>` a `<` sú volené tak, aby bolo na prvý pohľad vidieť o akú zmenu ide. Znak `>` znamená pridaný riadok, znak `<` znamená zmazaný riadok. Špecíalnym riadkom je oddelovač `---`, ktorý nájdeme v skupine zmenených riadkov. Oddelovač rozdeľuje skupinu na dve časti a tak určí, ktoré riadky majú byť nahradené druhými.

Kontextové formáty výstupu

Zvyčajne, keď pozeráme na rozdiely medzi súbormi, chceme vidieť nezmenené časti súborov blízko riadkov, ktoré sú rozdielne. Tieto blízke časti súborov voláme kontext.

`GNU diff` poskytuje dva formáty výstupu, ktoré zobrazia kontext okolo rozdielnych riadkov: kontextový formát výstupu a unifikovaný formát výstupu. Taktiež môže zobraziť, v ktorej funkcii alebo sekcii súboru sa nachádzajú rozdielne riadky.

Kontextový formát výstupu

Kontextový formát výstupu zobrazuje niekoľko riadkov kontextu okolo rozdielnych riadkov. Tento formát výstupu je štandardný pre distribúciu aktualizácií zdrojových kódov.

Kontextový formát výstupu začína s dvojriadkovou hlavičkou:

```
*** súbor1 čas-poslednej-zmeny-súboru1
--- súbor2 čas-poslednej-zmeny-súboru2
```

Ďalej následuje niekoľko skupín odlišných riadkov. Každá skupina zobrazuje jednu časť, kde sa súbory líšia. Napríklad:

```
***************
*** súbor1-rozsah-riadkov ****
  súbor1-riadok
  súbor1-riadok...
--- súbor2-rozsah-riadkov ----
  súbor2-riadok
  súbor2-riadok...
```

Riadky kontextu okolo riadkov, ktoré sa líšia, začínajú s dvoma medzerami. Riadky, ktoré sú rozdielne medzi súbormi, začínajú jedným z nasledujúcich znakov a medzerou, ktoré indikujú, o akú zmenu ide:

- `!` Riadok, ktorý je časťou skupiny jedného alebo viac riadkov, ktoré sú zmenené medzi dvoma súbormi. V druhom súbore je zodpovedajúca skupina riadkov označená `!`.
- `+` Riadok vložený do druhého súboru.
- `-` Riadok zmazaný z prvého súboru.

Unifikovaný formát výstupu

Unifikovaný formát výstupu je variácia kontextového formátu výstupu, ktorá je kompaktnejšia, pretože zanedbáva nadbytočné riadky kontextu.

Unifikovaný formát výstupu začína s dvojriadkovou hlavičkou:

```
--- súbor1 čas-poslednej-zmeny-súboru1
+++ súbor2 čas-poslednej-zmeny-súboru2
```

Následuje jeden alebo viac skupín odlišných riadkov. Každá skupina zobrazuje jednu časť, kde sa súbory líšia. Napríklad:

```
@@ súbor1-rozsah-riadkov súbor2-rozsah-riadkov @@
 riadok-každého-súboru
 riadok-každého-súboru...
```

Riadky bežne začínajú medzerou. Tie, ktoré sú v skutočnosti rozdielne, začínajú jedným z nasledujúcich znakov:

- `+` Riadok vložený do prvého súboru.
- `-` Riadok zmazaný z prvého súboru.

Formát výstupu vedľa seba

Formát výstupu vedľa seba je podobný normálnemu formátu výstupu, avšak zobrazené sú naraz oba súbory v dvoch stĺpcoch. V strede medzi stĺpcami sú značky:

- medzera Odpovedajúce si riadky sú rovnaké.
- | Odpovedajúce si riadky sú rozdielne.
- < Súbory sú rozdielne a iba prvý súbor obsahuje riadok.
- > Súbory sú rozdielne a iba druhý súbor obsahuje riadok.
- (Iba prvý súbor obsahuje riadok, ale rozdiel je ignorovaný.
-) Iba druhý súbor obsahuje riadok, ale rozdiel je ignorovaný.
- \ Odpovedajúce si riadky sú rozdielne a iba prvý riadok je neúplný.
- / Odpovedajúce si riadky sú rozdielne a iba druhý riadok je neúplný.

Ostatné formáty výstupu

Niektoré formáty výstupu produkujú skripty, ktoré dokážu vytvoriť z prvého súboru druhý. Podporované sú `ed` (editačné) skripty, spätné `ed` skripty a `RCS` skripty.

`GNU diff` podporuje aj porovnávanie zdrojových kódov jazyka C. Výstup takéhoto porovnania obsahuje všetky riadky z oboch súborov. Riadky spoločné pre oba súbory sa vo výstupe nachádzajú len raz, odlišné riadky sú oddelené pomocou preprocesoru jazyka C makrami `#ifdef name` alebo `#ifndef name`, `#else` a `#endif`. Pri preklade zdrojového kódu môžeme vybrať, ktorá verzia bude skompilovaná.

Ďalej môžeme použiť formát výstupu, ktorý špecifikuje označenie jednotlivých skupín textu. Môžeme nadefinovať špecifické značky, ktorými má zmenený text začínať a iné špecifické značky, ktorými má zmenený text končiť. Napríklad pri porovnaní TEXových súborov, môžeme určiť, že zmenené skupiny riadkov budú uzavreté do tagov `\begin{em}` a `\end{em}`. Ostatné riadky ponecháme bez zmien. Podobné úpravy značiek indikujúcich zmenený text môžeme nadefinovať aj pre jednotlivé riadky.

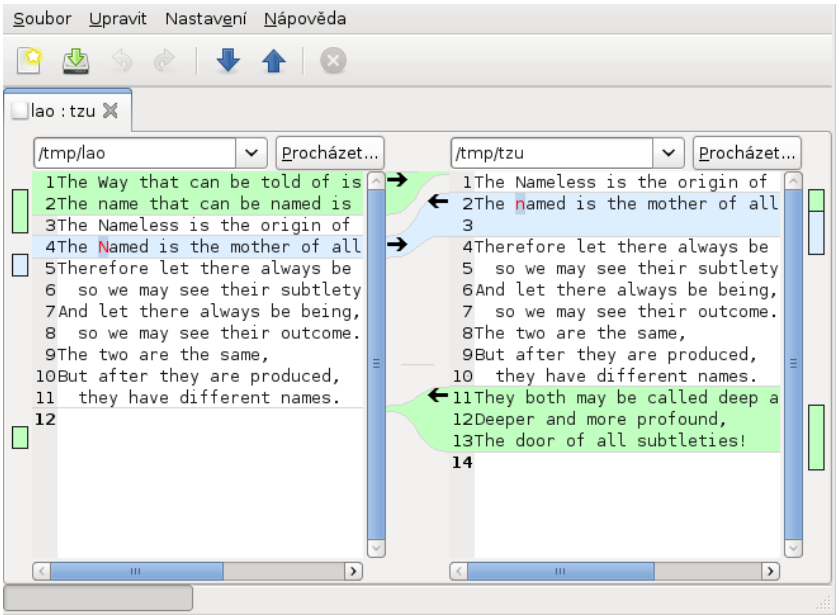
V tejto kapitole boli použité texty z [15] a [5].

3.2 Meld

Ako zástupcu programov na porovnávanie súborov s grafickým užívateľským rozhraním som zvolil program `meld`. Program `meld` dokáže prirodzene a prehľadne zobraziť rozdiely medzi dvomi alebo tromi textovými súbormi. Pomocou vstavaného textového editoru môžeme súbory priamo editovať a rozdiely budú dynamicky zobrazované. Okrem súborov vie program `meld` porovnávať aj celé zložky. Podobne ako existuje možnosť porovnávať dva alebo tri súbory, môžme porovnávať dve alebo tri zložky. Pri porovnávaní zložiek môžeme spustiť porovnávanie označených súborov. Veľmi výhodné rozšírenie je prechádzanie, prezeranie a upravovanie pracovných kópií z populárnych verzovacích systémov ako sú CVS, Subversion, Bazaar a Mercurial. Pri porovnávaní môžeme pomocou regulárnych výrazov nadefinovať, ktoré časti textu nemajú vplývať na porovnávanie, t.j. zmeny v nich budú ignorované.

Program `meld` je napísaný v programovacom jazyku Python. Grafické užívateľské rozhranie bolo vytvorené pomocou programu na konštrukciu grafického užívateľského rozhrania glade a knižníc GTK+.

Na obrázku 3.1 je vidieť ako prehľadne a intuitívne zobrazuje program `meld` zmeny medzi súbormi. Ďalšie ukážky z programu `meld` nájdeme v prílohe B.



Obrázek 3.1: Porovnávanie dvoch súborov pomocou programu meld

3.3 Ostatné programy na porovnávanie súborov

Okrem vyššie popísaných programov existuje ešte mnoho iných, určených na porovnávanie súborov. Tieto programy môžeme posudzovať podľa rôznych kritérií.

- rozhranie (grafické/konzolové)
- možnosti editoru (zvýrazňovanie syntaxe)
- formát záznamu rozdielov medzi súbormi (XML,UNIX diff, HTML, CSV)
- podpora vytvárania editačných skriptov
- zobrazenie zmien v riadku
- porovnávanie adresárov
- trojcestné porovnávanie
- porovnávanie bajt po bajte
- zlučovanie dvoch rozdielnych súborov
- platforma (Linux, Windows, Macintosh)
- cena programu a licencia (GPL, voľná, proprietárna)

Tieto jednotlivé programy porovnávajú vždy len špecifické typy súborov. Najčastejšie sú zamerané na porovnávanie textových súborov riadok po riadku. V praxi ale treba porovnávať aj iné typy. Takýto program, ktorý porovnáva rôzne typy súborov si neskôr navrhneme a implementujeme.

Kapitola 4

Typy dokumentov

V tejto kapitole uvedieme jednotlivé typy dokumentov, ktoré bude program `mdiff` porovnávať. Opíšeme štruktúru každého dokumentu, vysvetlíme si jeho charakteristické znaky a náležitosti.

4.1 Textový súbor

Textový súbor je jeden zo základných súborových formátov používaných v počítačoch. Slúži na uloženie prostého textu pomocou sekvencie znakov. Súbor nie je nijak štrukturovaný a neobsahuje špeciálne formátovacie značky. Textový súbor sa najčastejšie používa pri strojovom spracovaní textu alebo tam, kde nie je podstatné vizuálne formátovanie podaného textu, ale jeho obsah.

4.1.1 Formát

V bežnom textovom súbore sú znaky reprezentované pomocou ASCII tabuľky, kde jeden znak zodpovedá jednému bajtu. Okrem viditeľných znakov obsahuje textový súbor nasledujúce „biele“ znaky (medzera, tabulátor, znak nového riadku).

4.1.2 Kódovanie

Veľký problém, týkajúci sa textových súborov, je použité kódovanie znakov. Rôzne krajiny používajú rôzne znaky a pri ukladaní znakov na ôsmich bitoch nedokážeme zakódovať všetky znaky národných abeced. Čiastočné riešenie bolo zavedenie rôznych kódovaní, kde sa vrchná polovica ASCII tabuľky doplnila znakmi národnej abecedy. Pre Strednú Európu platí štandardizované kódovanie ISO 8859-2 a kódovanie CP1250 firmy Microsoft. Pri práci s textovým súborom je potrebné vedieť, v akom kódovaní je text súboru uložený, aby sme mohli súbor korektne zobraziť. Vyriešenie problému s rôznymi kódovaniami pomohlo zavedenie jednotného štandardného kódovania Unicode obsahujúceho všetky národné abecedy. Najčastejšie používané kódovanie je UTF-8. Nevýhodou Unicodu je väčší objem dát voči rovnakému textu v ASCII.

4.2 LaTeXový súbor

LaTeX je nadstavbou nad systémom TeX a je to profesionálny typografický systém určený na sadzbu textov. Napríklad aj táto práca bola tiež vytvorená pomocou systému LaTeX. Pri

sadzbe textu si vytvoríme zdrojový súbor, ktorý bude obsahovať text práce a formátovacie značky. Tento zdrojový súbor následne preložíme jedným z prekladačov programu LaTeX.

4.2.1 Formát značiek

LaTeXový súbor obsahuje text práce a formátovacie značky, ktoré definujú vzhľad výsledného dokumentu. Formátovacích značiek je viac než tristo a samotný užívateľ si môže vytvárať nové. Značky majú presne definovanú syntax a jednoznačný význam. Obecne môžeme popísať formátovaciu značku nasledovne: `\príkaz[]{}`, kde `príkaz` značí samotný príkaz, v zložených zátvorkách sa môže vyskytovať v závislosti na príkaze text, prostredie alebo parametre. V hranatých zátvorkách sa môžu vyskytovať parametre.

4.2.2 Štruktúra dokumentu

Samotný dokument sa skladá z dvoch hlavných častí:

- preambula
- textová časť

Preambula začína príkazom `\documentclass[voľby]{trieda}[dátum vytvorenia]` a pokračuje až po začiatok textovej časti dokumentu. V preambuli sú príkazy definujúce typ dokumentu, použitý formát, použité balíčky príkazov a iné. V preambuli tiež môžeme definovať vlastné príkazy a uviesť delenie slov, ak systém nevie niektoré slová na konci riadku rozdeliť.

Za preambulou nasleduje textová časť uzatvorená do príkazov `\begin{document}` a `\end{document}`. V textovej časti je už samotný text práce s formátovacími značkami.

4.3 Konfiguračný súbor

Konfiguračný súbor je textový súbor s určitými špecifickými príkazmi. Konfiguračný súbor sa používa na uloženie nastavení – konfigurácie jednotlivých aplikácií, servrov, poprípade celého operačného systému.

4.3.1 Syntax

Syntax konfiguračných súborov nie je presne špecifikovaná, ale existujú určité pravidlá, ktoré sa dodržujú pri vytváraní konfiguračných súborov. Konfiguračné súbory sú písané v ASCII kódovaní, zriedka v UTF-8 a sú riadkovo orientované – čo riadok, to jednotlivá položka. To vytvára jednoduchú databázu nastavení.

V súčasnosti sa upúšťa od riadkovo orientovaných konfiguračných súborov a prechádza sa na konfiguračné súbory uložené vo formáte XML, poprípade YAML. Tieto formáty majú mnoho výhod ako, napríklad, presne definovanú syntax, mnoho nástrojov na skontrolovanie validity syntaxe, verifikáciu syntaxe alebo jej zvýraznenie.

4.3.2 Použitie

Programy načítajú nastavenia z konfiguračného súboru pri ich spustení, periodicky ich kontrolujú a pri zmene nastavenia si ich program znovu načíta a aplikuje zmeny na súčasný proces.

4.3.3 UNIXové systémy

V UNIXových operačných systémoch existujú stovky rôznych formátov konfiguračných súborov. Každá aplikácia, démon alebo server, môže mať unikátny formát konfiguračného súboru. Kedysi dávno sa dal operačný systém UNIX nastavovať len pomocou editácie týchto súborov.

V UNIXovom konfiguračnom súbore, riadok začínajúci znakom # znamená komentár.

Užívateľské aplikácie často vytvárajú súbor alebo zložku v domovskej zložke užívateľa na uloženie svojej konfigurácie. Takto má každý užívateľ uložené svoje nastavenia aplikácie.

Systémové konfiguračné súbory majú svoje miesto v zložke `/etc`.

4.3.4 Microsoft Windows

V OS Windows používajú aplikácie na ukladanie svojej konfigurácie registre systému Windows alebo INI konfiguračné súbory [8].

4.4 Open Document Format

Open document format (ODF) je otvorený súborový formát určený k ukladaniu a výmene dokumentov vytvorených kancelárskymi aplikáciami. ODF pokrýva textové dokumenty, prezentácie, tabuľky, grafy a databázy. Štandard ODF bol vyvinutý združením OASIS a je založený na XML [12].

4.4.1 Popis ODF

ODF obsahuje určitú adresárovú štruktúru, v ktorej sú uložené XML a iné binárne súbory. Táto adresárová štruktúra je následne skomprimovaná do ZIP archívu, čím podstatne zredukujeme objem ukladaných dát. V nasledujúcom texte si popíšeme Open Document Text – formát súboru na ukladanie textu.

V archíve ODT súboru sa nachádzajú tieto hlavné súbory:

mimetype – textový súbor obsahujúci MIME typ, pre súbor ODT je mime typ `application/vnd.oasis.opendocument.text`

meta.xml – XML súbor, ktorý obsahuje informácie o súbore ako čas vytvorenia, login užívateľa, ktorý dokument vytvoril, názov a verziu programu, cez ktorý sme dokument vytvorili ...

settings.xml – nastavenie aplikácie, v ktorej bol dokument vytvorený

styles.xml – štýly použité v dokumente

content.xml – samotný obsah dokumentu (text) a automatické štýly

4.4.2 Obsah súboru content.xml

V súbore content.xml sa nachádzajú skripty, deklarácie fontov, automatické štýly a hlavne samotný text dokumentu. Text je uzatvorený spolu s formátovacími značkami v elemente `<office:body>`. Samotný text obsahuje elementy `<text:span>`, `<text:p>` a `<text:h>`. Význam tagov je podobný ako pri HTML. Text v tagu `<text:span>` sa bude zobrazovať ako riadkový element, text v tagoch `<text:p>` a `<text:h>` sa bude zobrazovať ako blokový

element. Blokový element znamená, že za takýmto textom bude prechod na nový riadok, za riadkovým elementom prechod na nový riadok nebude. Pomocou atribútov v týchto tagoch môžeme formátovať text. Atribút `text:style-name` berie ako hodnotu názov definovaného štýlu.[11]

Text často formátujeme pomocou rôznych pomôcok ako napríklad číslovaných/nečíslovaných odrážok, tabuliek alebo iného formátovania. Na tieto pomôcky sú určené špeciálne elementy. Vzhľadom na typ tejto práce nebudeme dopodrobna uvádzať jednotlivé elementy a čitateľa odkážeme na špecifikáciu ODF [11].

4.5 Grafické formáty

4.5.1 Bitová mapa

Bitová mapa (BMP) je súborový formát na ukladanie rastrovej grafiky. Veľkou výhodou tohoto formátu je jeho jednoduchosť a to, že nie je chránený patentom. Obrázky BMP sú ukladané po jednotlivých pixeloch. Súbory vo formáte BMP väčšinou nepoužívajú žiadnu kompresiu. V praxi sa pre bezstratové ukladanie obrázkov používajú viac novšie formáty ako PNG alebo TIFF [7].

4.5.2 PBM

Grafický formát PBM je jedným z troch formátov:

PPM je formát na uloženie farebného obrázku.

PGM je formát na uloženie čiernobieleho obrázku pomocou odtieňov šedej farby.

PBM je formát na uloženie čiernobieleho obrázku pomocou dvoch farieb.

Tieto formáty boli navrhnuté na výmenu grafických dát, sú veľmi jednoduché a otvorené. Ich veľkou nevýhodou je ale výsledná veľkosť takéhoto súboru. Pri ukladaní si môžeme vybrať medzi dvoma variantami: textovou a binárnou. V nasledujúcom texte si ukážeme formát PBM, jeho binárnu formu a textovú.

Ako ukážku si predvedieme obrázok písmena „J“ v binárnom a textovom formáte PBM:

```
                P1
                # subor J.PBM
                6 10
000010          0 0 0 0 1 0
000010          0 0 0 0 1 0
000010          0 0 0 0 1 0
000010          0 0 0 0 1 0
000010          0 0 0 0 1 0
000010          0 0 0 0 1 0
100010          1 0 0 0 1 0
011100          0 1 1 1 0 0
```

Obrázok pozostáva z dvojrozmerného poľa, o rozmeroch obrázku, kde je každý pixel reprezentovaný jedným bitom. Ak má tento pixel čiernu farbu, je tento bit nastavený do jednotky, inak do nuly. Výsledný čiernobiely obrázok má potom veľkosť rovnú jeho rozmerom.

Textový formát je veľmi podobný binárnemu, ale obsahuje naviac hlavičku a pre každý pixel náleží jeden ASCII znak, vyjadrujúci farbu tohto pixelu.

Štruktúra súboru je následovná:

- Prvá položka je „magické“ číslo identifikujúce typ súboru. Pre čiernobiely PBM formát je to P1.
- V hlavičke sa môžu vyskytovať komentáre – riadky začínajúce znakom „#“ a končiace znakom nového riadku.
- Ďalšou položkou hlavičky je rozmer obrázku, v našom prípade 6×10.
- Posledný parameter hlavičky udáva maximálnu hodnotu každého pixelu. Udáva sa pri formátoch PGM (stupne šedi) a PPM (farebný).
- Za hlavičkou nasleduje dvojrozmerné pole pixelov o veľkosti rozmerov obrázku, kde je každý pixel vyjadrený číslom vo forme ASCII znakov. Medzi jednotlivými znakmi sa môžu vyskytovať medzery.

Rozdiel medzi formátmi PBM, PGM a PBM je len v tom, aké hodnoty môže jednotlivý pixel mať. Pri čiernobielom PBM formáte sú povolené hodnoty `0` a `1`. Pri formáte PGM používajúcom odtiene šedej farby môže každý pixel dosahovať hodnotu uvedenú ako posledný parameter v hlavičke. Nakoniec, pri farebnom formáte PPM je každý pixel uložený ako trojica čísel vyjadrujúca farebné zložky R G B alebo ako jedno číslo vyjadrujúce farbu pixelu. Maximálna hodnota tohoto čísla (trojice čísel) je uvedená ako posledný parameter hlavičky súboru.[1]

4.5.3 JPEG

JPEG je grafický formát používajúci stratovú kompresiu. Stupeň kompresie môžeme nastaviť tak, aby sme dosiahli ideálny pomer medzi objemom dát a kvalitou obrázku. Často dosahujeme pomer 10:1 s malou stratou kvality. JPEG je dnes jeden z najpoužívanejších formátov na ukladanie obrázkov a fotografií.

4.5.4 PNG

PNG je grafický formát určený na bezstratovú kompresiu rastrovej grafiky. Bol vyvinutý ako náhrada za GIF. Podporuje väčšiu hĺbku farieb, obsahuje osembitovú priehľadnosť – alfa kanál a má lepší kompresný algoritmus. [13]

Kapitola 5

Analýza a návrh aplikácie

Pri vytváraní programu `mdiff` bola venovaná dostatočná pozornosť podrobnej analýze problému a kvalitnému návrhu riešenia.

Pri vývoji programu bol použitý iteračný model. Jeden cyklus iteračného modelu pozostáva z nasledujúcich etáp:

1. špecifikácia
2. návrh
3. implementácia
4. testovanie

Jednotlivými etapami sa prechádza v uvedenom poradí. Ak zistíme chybu počas vývoja aplikácie, cyklus opakujeme. Takýmto opakovaním eliminujeme väčšinu chýb a nedostatkov.

5.1 Špecifikácia programu

Zadanie programu je uvedené na začiatku tejto práce.

Po konzultácii s vedúcim práce sme sa dohodli na vytvorení programu `mdiff` (Multiple document type DIFF), ktorý bude implementovať nasledujúce moduly:

TextDiff – modul, ktorý má rovnakú funkcionalitu ako súčasný program `diff`.

FileInfo – modul, ktorý zistí informácie o súboroch ako veľkosť, dátum a čas vytvorenia poslednej modifikácie, meno vlastníka a iné. Tieto údaje porovná a zobrazí rozdiel.

LatexDiff – modul, ktorý vyextrahuje z latexových súborov čistý text bez formátovacích značiek, tieto texty porovná a následne zobrazí rozdiel.

ConfigFileDiff – modul, ktorý načíta dva konfiguračné súbory, porovná ich a zobrazí rozdiel medzi nimi. Poradie riadkov nie je dôležité.

OpenOfficeDiff – modul, ktorý z dvoch súborov vo formáte ODT (Open Document Text) vyextrahuje text, ten porovná a následne zobrazí rozdiel.

ImageDiff – modul, ktorý porovná dva obrázky a zobrazí, v čom sa líšia.

Niektoré moduly nebolo nutné implementovať. XMLDiff – modul, ktory porovnáva XML súbory, VideoDiff – modul na porovnávanie videí a nakoniec AudioDiff – modul, porovnávajúci audio súbory. XMLDiff bol vylúčený preto, že už existuje aplikácia, ktorá porovnáva XML dokumenty. Táto aplikácia ma rovnomenný názov `xmldiff`, má množstvo rôznych volieb a nastavení, preto bola jej reimplementácia odložená. V pokračovaní tohto projektu môže byť `xmldiff` analyzovaný a pripojený cez určité rozhranie k programu `mdiff`.

Iná situácia nastala pri moduloch VideoDiff a AudioDiff. Tieto dva moduly si vyžadujú oveľa väčší priestor na realizáciu, než poskytuje tento projekt, preto sme sa rozhodli implementácie týchto dvoch modulov nechať ako možné budúce rozšírenie programu `mdiff`.

Keďže program `diff` je konzolová aplikácia, rozhodli sme sa, že jeho nástupca, program `mdiff`, bude takisto konzolová aplikácia, bez grafického užívateľského rozhrania. Všetky výstupy z programu `mdiff` budú textové, aby sa dali ďalej strojovo spracovávať. Zo zadania vyplývajúca modularita aplikácia umožní budúce rozširovanie a zdokonaľovanie programu alebo použitie jednotlivých modulov ako knižníc v iných programoch.

5.2 Vývojové prostriedky

Pre úspech aplikácie hrajú veľmi dôležitú úlohu faktory ako platforma, na ktorej aplikácia pobeží a s ňou spojená prenositeľnosť programu, rýchlosť samotnej aplikácie, závislosť na inom softvéri a rýchlosť vývoja aplikácie – uvedenie na trh. Tieto hlavné faktory vo veľkej miere ovplyvňuje zvolený programovací jazyk, v ktorom je aplikácia naprogramovaná, spôsob programovania a vybrané knižnice, ktoré aplikácia používa. Na dosiahnutie optimálneho pomeru medzi uvedenými faktormi vhodne zvoliť vývojové prostriedky ako

- programovací jazyk
- použité knižnice
- vývojové prostredie a
- systém pre správu a verzovanie zdrojových kódov.

Pri výbere hlavnej časti vývojových prostriedkov – programovacieho jazyka, boli kladené tieto požiadavky:

Prenositeľnosť – zdrojové kódy musia byť použiteľné na čo najväčšom počte architektúr a operačných systémov.

Rýchlosť – program musí byť dostatočne rýchly aj s väčším objemom spracovávaných dát.

Knižnice – podporujú efektivitu tvorby programu, taktiež musia byť na všetkých požadovaných platformách.

Efektivita – rýchlosť tvorby zdrojových kódov.

5.2.1 Voľba programovacieho jazyka

Pri výbere programovacieho jazyka sa medzi favoritov dostali C, C++, Java a Python. V nasledujúcich riadkoch uvedieme jednotlivé klady a zápory každého z vybraných programovacích jazykov.

C

Jazyk C je procedurálny kompilovaný programovací jazyk, ktorý vyniká svojou rýchlosťou. Pri zachovaní noriem má veľmi dobrú podporu na rôznych operačných systémoch. V jazyku C sa tiež dajú tvoriť modulárne aplikácie. Pre tento jazyk existuje mnoho nástrojov podporujúcich vývoj aplikácií.

Nevýhodou je slabá efektivita tvorby kódu, pretože C je procedurálny a nie objektovo orientovný programovací jazyk, ponúka len základné dátové typy a minimum štandardných knižníc.

Táto aplikácia by sa dala naprogramovať pomocou jazyka C, ale vyžadovalo by to zbytočne veľa prostriedkov.

C++

C++ je na rozdiel od jazyka C objektovo orientovaný kompilovaný programovací jazyk, ktorý má už slušnú podporu štandardných knižníc. Rýchlosť výslednej aplikácie je dobrá. S prenositeľnosťou je to podobne ako pri Cčku. Pre C++, podobne ako pre jazyk C, existuje mnoho nástrojov podporujúcich vývoj aplikácií.

Program `mdiff` by sa dal naprogramovať pomocou jazyku C++, pretože ponúka vhodné prostriedky, či už objektovú orientáciu, podporu knižníc alebo modularitu.

Java

Java je objektovo orientovaný interpretovaný jazyk. Má obrovskú podporu knižníc, výbornú prenositeľnosť vďaka interpretácii bajtkódu virtuálnym strojom, ktorý je dostupný na väčšine platforiem a veľa prostriedkov pre profesionálny vývoj aplikácií.

Nevýhody Javy sú jej rýchlosť, nakoľko je Java interpretovaný jazyk, je výrazne pomalšia, než kompilované jazyky. Ďalšou veľkou nevýhodou je závislosť programu na prítomnosti virtuálneho stroja.

Vytvorenie programu `mdiff` v jazyku Java je možné. Programovanie by vyžadovalo minimum prostriedkov vďaka podpore knižníc, ale rýchlosť by bola výrazne nižšia voči napríklad C++.

Python

Python je moderný dynamický objektovo orientovaný interpretovaný programovací jazyk. Tento jazyk disponuje veľkým počtom štandardných knižníc určených na rôzne účely. Vďaka tomu, že tento jazyk je dynamicky vyvíjaný oproti napríklad C/C++, ktorých posledné štandardy majú niekoľko rokov a veľkej podpore knižníc, ktoré sa pravidelne dopĺňajú novými, aplikácie programované v Pythone sa vyvíjajú veľmi rýchlo a efektívne. Tento jazyk má interpret napísaný v jazyku C, takže interpretácia je dostatočne rýchla, poprípade pre náročné výpočty môžeme daný modul naprogramovať v jazyku C/C++, skompilovať a následne ho používať. Interpret jazyka Python je implicitne nainštalovaný vo väčšine operačných systémov GNU/Linux, pre ostatné platformy je zdarma k dispozícii inštalačný balíček.

Nevýhodou jazyku Python je jeho rýchlosť, nakoľko to je interpretovaný jazyk. Tento problém sa dá, ale ľahko obísť tým, že problematické úseky kódu naprogramujeme v jazyku C/C++. V tomto čase sa taktiež rozbieha projekt na optimalizáciu interpretu jazyka Python. Tento projekt ma ambiciózny cieľ, urýchliť interpret jazyka Python až 5-krát. V čase

implementácie programu `mdiff` vyšla dlho očakávaná verzia Pythonu – Python 3.0, ktorá prináša viacero zmien.

Implementácia programu `mdiff` pomocou jazyka Python sa javí ako najvhodnejšie možné riešenie. Tento jazyk sa v poslednej dobe stal veľmi obľúbený, a to hlavne pre svoju jednoduchosť, efektívnosť tvorby kódu a obrovskú podporu knižníc. Nemalou výhodou je prítomnosť knižnice difflib, ktorá priamo implementuje triedy a funkcie na porovnávanie reťazcov.

5.2.2 Knižnice

Ďalšou kľúčovou zložkou úspešného návrhu aplikácie je nájsť a naštudovať vhodné knižnice, ktoré budeme následne používať pri implementácii. Keďže sme ako implementačný jazyk zvolili Python, preskúmame aké štandardné knižnice jazyk ponúka. Pri používaní štandardných knižníc jazyka Python máme istotu prenositeľnosti aplikácie. Zo štandardných knižníc budeme používať nasledujúce:

- **string, re** na prácu s reťazcami
- **difflib** na porovnávanie reťazcov
- **os, stat, pwd, grp** na zistenie informácií o súbore
- **optparse** na parsovanie argumentov príkazového riadku
- **mimetypes** na zistenie typu súboru
- **zipfile** na rozbalenie obsahu ODT súboru

Jednotlivé knižnice a prácu s nimi budeme popisovať pri ich použití.

5.2.3 Vývojové prostredie a správa zdrojových kódov

Pred samotným programovaním musíme ešte vhodne zvoliť, v akom vývojovom prostredí budeme program vytvárať a aký prostriedok použijeme na správu a verzovanie zdrojových kódov.

Vývojové prostredie

Na samotné programovanie potrebujeme iba textový editor, v ktorom budeme písať zdrojový kód aplikácie a interpret jazyka Python. Táto kombinácia je postačujúca, ale nie produktívna.

Vývojové prostredie, v ktorom bol program `mdiff` naprogramovaný bolo *geany* [9]. Pre toto IDE som sa rozhodol preto, lebo s ním mám veľmi dobré skúsenosti. Geany je malé, jednoduché a prehľadné multiplatformné IDE so základnou funkcionalitou, ktorú programátor potrebuje.

Správa zdrojových kódov

Na správu a verzovanie zdrojových súborov bol použitý verzovací systém SVN [14]. Repozitár na ukladanie verzií som vytvoril na školskom servri Merlin na svojom účte. Po prihlásení lokálnej pracovnej zložky stačilo už len pravidelne ukladať na server zmeny. Ako klienta na komunikáciu s SVN repozitárom som použil program *RapidSVN* – jednoduchý SVN klient s dostatočnou funkcionalitou.

5.3 Dekompozícia problému

Teraz môžeme pristúpiť k samotnej analýze a následnému návrhu programu. Pri návrhu programu použijeme metódu dekompozície.

Máme navrhnúť program, ktorý bude porovnávať rôzne typy súborov. Tento všeobecný problém si rozdelíme na menšie, špecifickejšie podproblémy. Nakoľko Python podporuje modulárne programovanie, pokúsime sa každý podproblém implementovať pomocou samostatného modulu.

5.3.1 Spustenie programu

Program sa bude spúšťať nasledovne: `mdiff [voľby] prvý_súbor druhý_súbor`

Aby sme mohli vyšetriť argumenty príkazového riadku, musíme si najskôr určiť, aké voľby bude program mať a aký bude ich význam. Povinná voľba musí byť samozrejme nápoveda (help) `-h`, respektíve `--help`. V nápovede sa uvádza použitie, jednoduchý popis programu, verzia a všetky parametre a ich význam. Ďalšie parametre si budeme definovať podľa potreby.

Pre jednoduchosť používania programu by bolo vhodné, aby program sám rozpoznal o aký typ súboru sa jedná (obrázok, text, odt súbor, ...) a zavolal adekvátnu funkciu. To sa dá vyriešiť pomocov MIME typov súborov. Ak program nedostane žiadne parametre, určujúce typ porovnávaných súborov, pokúsi sa typ zistiť sám.

5.3.2 Knižnica na porovnávanie

Jadrom programu, ktorý ma porovnávať súbory, bude modul `difflib2`, ktorý poskytuje funkcionalitu a algoritmy na takéto porovnávanie. Názov difflib2 nie je náhodný, je to pôvodná štandardná knižnica `difflib` programovacieho jazyku Python, ktorú sme museli pre potreby programu `mdiff` mierne upraviť a rozšíriť o novú funkcionalitu. Táto nová vylepšená knižnica poskytuje funkcie na porovnávanie dvoch sekvencií. Pre našu potrebu budeme za sekvencie pokladať zoznam reťazcov textu. `Difflib2` dokáže porovnať dva zoznamy reťazcov, nájsť odlišnosti a vrátiť ich v rôznych formátoch. Medzi primitívne formáty patrí zoznam indexov rovnakých častí, zoznam indexov so značkami zmazaný, rovnaký a vložený. Zo zložitejších výstupných formátov spomeňme napríklad kontextový formát výstupu, popísaný na strane 14, unifikovaý formát výstupu, uvedený na strane 15 alebo formát výstupu ndiff.

Ak má program `mdiff` podporovať rovnaké voľby, napríklad ignorovanie veľkosti znakov alebo počtu medzier, ako program `diff`, musí toto podporovať aj knižnica, ktorú budeme používať na porovnávanie dvoch sekvencií. Štandardná knižnica `difflib` by dokázala porovnať dve sekvencie, ale veľký problém nastáva, ak by sme chceli pri porovnaní ignorovať veľkosť písmen alebo medzery. Táto možnosť v tejto knižnici nebola. Preto sme museli rozšíriť funkcionalitu štandardnej knižnice.

5.3.3 Porovnávanie textov

Porovnávanie textov je jednou z hlavných častí práce, pretože až na obrázky budú všetky typy súborov reprezentované pomocou textu. Pri spracovávaní textu narazíme na niektoré problémy, popísané na strane 18 ako použité kódovanie alebo znaky konca riadkov.

Problém so znakmi konca riadkov rieši samotný jazyk Python. Ako použité kódovanie sme zvolili medzinárodné kódovanie UTF-8. Toto kódovanie je pre jazyk Python 3.0 a jeho

knižnice implicitné. Preto ak budeme chcieť porovnávať texty v inom kódovaní, budeme musieť texty prekódovať, napríklad programom `iconv`, do UTF-8.

Ignorovanie veľkosti písmen a počtu medzier v riadku

Navrhované riešenie vychádza z predpokladu, že funkcia porovnáva text po riadkoch a vracia zoznam indexov rovnakých riadkov. Implementácia samotného algoritmu pre nájdenie najdlhšej spoločnej subsekvencie riadkov je v knižnici `difflib` implementovaná pomocou asociatívneho poľa – v Pythone dátová štruktúra dictionary. Riadky textu sú použité ako kľúč a hodnota položky je zoznam indexov, kde sa riadok nachádza. Keďže na porovnávanie dvoch reťazcov nie je použitá nejaká funkcia, ktorú by sme mohli upraviť, ale asociatívne pole a jeho vyhľadávacie mechanizmy (hashovanie), budeme musieť problémy ignorovania veľkosti písmen a počtu medzier riešiť inak než zmenou takejto funkcie.

Riešenie je nasledovné: Vytvoríme si v pamäti pozmenenú kópiu oboch textov. To znamená, pri ignorovaní veľkosti písmen budú obe nové kópie obsahovať všetky písmená rovnakej veľkosti, pri ignorovaní bielych znakov zredukujeme v oboch súboroch nadbytočné medzery a tabulátory na jednu medzeru. Takto dosiahneme cielený výsledok za cenu väčšej pamäťovej náročnosti, avšak program bude rýchlejší, pretože nebude musieť pri každom porovnávaní riadkov daný riadok upravovať. Všetky zmeny, ktoré prevedieme v jednom riadku, nám neovplyvnia výsledné indexy. Tieto indexy sa budú zhodovať s originálnym neupraveným textom, ktorý budeme vypisovať. Takto môžeme rozšíriť spomenuté štandardné voľby o ďalšie ako ignorovanie časti riadku zodpovedajúcej regulárnemu výrazu.

Ignorovanie riadkov

Pri ignorovaní riadkov je situácia diametrálne odlišná. Ak by sme chceli ignorovať napríklad prázdne riadky ich vymazaním a následným porovnaním, výsledné indexy nebudú korešpondovať s originálnym textom. Jediné prijateľné riešenie je riadky vymazať priamo pri načítavaní textu. Toto riešenie však neumožní zobraziť správne indexy s originálnym súborom a je vhodne hlavne na kontrolu rozdielov človekom.

Reimplementácia funcionality programu `diff`

Program `diff` sme si popísali na strane 12. Náš program `mdiff` má nahrádzať tento program `diff`, preto musíme reimplementovať všetky hlavné voľby programu `diff`. V našom prípade to budú tieto:

`-i, --ignore-case` ignoruje zmeny vo veľkosti písmen

`-b, --ignore-space-change` ignoruje zmeny v počte medzier a tabulátorov

`-w, --ignore-all-space` ignoruje zmeny v počte medzier, tabulátorov a prázdnych riadkov

`-B, --ignore-blank-lines` ignoruje zmeny v počte prázdnych riadkov

`-I RE, --ignore-matching-lines=RE` ignoruje zmeny spôsobené vložením alebo zmazaním riadku odpovedajúcemu regulárnemu výrazu `RE`

Rovnako ako musíme reimplementovať voľby programu `diff`, musíme reimplementovať aj jeho formáty výstupu, popísané na strane 13. Kontextový a unifikovaný formát výstupu

je priamo implementovaný v knižnici `difflib`, ale normálny formát výstupu musíme doimplementovať. Knižnica `difflib` ponúka naviac svoje ďalšie dva formáty výstupu, a to ndiff popísaný v prílohe A.6 a výstupu v html – príloha A.7.

Ndiff formát je veľmi vhodný pre prehliadanie človekom. Celý text je zobrazený a odlišné riadky začínajú znakom `+` alebo `-`.

5.3.4 Porovnávanie informácií o súbore

Informácie o súbore, ktoré si uschováva OS sa mierne líšia systém od systému. V našom prípade budeme používať na získanie týchto informácií štandardné knižnice jazyka Python, čím si zabezpečíme prenositeľnosť. O súbore môžeme zistiť tieto údaje:

Meno – meno súboru.

Typ – typ súboru, v UNIXových OS poznáme tieto typy: zložka, špeciálny blokový súbor, špeciálny znakový súbor, normálny súbor, frontu fifo, symbolický link a soket.

Vlastník – vlastník súboru, môžeme vyjadriť menom alebo číslom.

Skupina – skupinu, do ktorej vlastník súboru patrí, môžeme vyjadriť menom alebo číslom.

Prístupové práva – v UNIXových OS máme tri skupiny – vlastník, skupina, ostatní – ktorým môžeme priradiť práva čítania, zápisu alebo spúšťania.

Veľkosť – veľkosť súboru.

Čas – údaje o čase sa líšia podľa OS. V UNIXových OS sa uchovávajú dva údaje – čas posledného prístupu k súboru a čas poslednej modifikácie súboru.

Pre každý súbor zistíme všetky dostupné informácie a zobrazíme rozdiel v ndiff formáte.

5.3.5 Porovnávanie konfiguračných súborov

Špecifikáciu konfiguračného súboru sme si uviedli na strane 19. V jednoduchosti povedané, konfiguračný súbor je riadkovo orientovaný textový súbor, kde sú na jednotlivých riadkoch uložené voľby, ale na ich poradí nezáleží. Keďže nezáleží na poradí riadkov, nemôžeme použiť knižnicu `difflib`.

Najjednoduchšie riešenie je načítať oba súbory po riadkoch, prvý ako zoznam riadkov, druhý do asociatívneho poľa, kde kľúč bude samotný riadok a hodnota bude zoznam indexov, kde sa tento riadok nachádza. Pri načítavaní súborov budeme ignorovať komentáre – riadky začínajúce znakom #. Porovnávanie bude potom prebiehať nasledovne: budeme prechádzať po riadkoch prvého súboru a pokúsime sa pre každý riadok nájsť rovnaký riadok v asociatívnom poli riadkov druhého súboru. Ak tam riadok je, začleníme ho do skupiny rovnaké a znížime počet indexov (hodnota položky asociatívneho poľa) o jednu. Ak sa tam riadok nenachádza, priradíme tento riadok do skupiny zmazané. Nakoniec prejdeme zvyšné položky v asociatívnom poli a tieto riadky budú tvoriť skupinu pridaných. Takto nám vznikli tri skupiny riadkov, ktoré už len stačí vypísať vo vhodnom formáte. Keďže nezáleží na poradí riadkov, zvolíme formát, ktorý neuvádza indexy – ndiff formát.

5.3.6 Porovnávanie latexových súborov

Latexový súbor, ako sme si ho popísali na strane 18, obsahuje formátovacie značky a samotný text práce. My potrebujeme z oboch súborov vyextrahovať práve samotný text bez značiek a ten následne porovnať.

Ponúkajú sa nám dva možné spôsoby ako takýto súbor spracovávať: pomocou latexového parseru alebo pomocou regulárnych výrazov. Pretože potrebujeme vyextrahovať text a nezaujímajú nás významy jednotlivých formátovacích značiek, zvolíme druhú možnosť – regulárne výrazy. Nadefinujeme si potrebné regulárne výrazy, ktoré nám vymažú formátovacie značky a v pamäti nám zostane len čistý text. Takýto text môžeme ľahko porovnať a zobraziť rozdiel. Ako formát výstupu sme zvolili ndiff, pretože latex sám určuje výsledný vzhľad dokumentu – zalomenie riadkov. Ndiff je pre človeka lepšie čitateľný a nezobrazuje indexy, ale celý text.

5.3.7 Porovnávanie súborov Open Office Text

Súbor ODT je podľa popisu zo strany 20 niekoľko XML súborov skomprimovaných do jedného archívu. Súbor, ktorý nás bude najviac zaujímať je content.xml, pretože práve on obsahuje samotný text práce. Naším cieľom je vyextrahovať z dvoch ODT súborov text, ten porovnať a zobraziť rozdiel.

V prvom rade musíme z archívu rozbaliť a načítať do pamäti súbor content.xml. K tomuto XML súboru môžeme pristupovať pomocou DOM. Pre nás je ale podstatný iba samotný text a nezaujímajú nás formátovacie tagy. Zo špecifikácie vieme, že text je uzatvorený v tagoch `<text:span>`, `<text:p>` a `<text:h>`, preto stačí použiť regulárne výrazy na extrakciu textu z týchto tagov. Toto riešenie je aj výpočtovo efektívnejšie a má menšiu pamäťovú aj časovú náročnosť. Takto vyextrahované texty porovnáme a zobrazíme rozdiel. Použitý formát výstupu bol ndiff, pretože tak ako v latexe, aj v Open Office výsledný vzhľad dokumentu – zalomenie riadkov – počíta sám program. Ndiff je pre človeka lepšie čitateľný a nezobrazuje indexy, ale celý text. Štandardne je na riadku zobrazených 80 znakov, ale túto voľbu si môžeme sami nastaviť.

5.3.8 Porovnávanie obrázkov

Obrázok, ako sme si popísali na strane 21, je dvojrozmerné pole pixelov. Každý pixel má farbu, vyjadrenú troma farebnými zložkami. Pre nás najvhodnejšie bude otvoriť si oba obrázky a prechádzať po jednotlivých pixeloch a kontrolovať, či sú totožné.

Grafický výstup

Musíme si vhodne zvoliť formát, v ktorom budeme zmeny vyjadrovať. Keďže naša aplikácia má mať textový výstup, javí sa ako najvhodnejšia možnosť použiť formát PBM popísaný na strane 21. Tento formát je textový a dá sa zobraziť priamo v konzole alebo uložiť do súboru, ktorý môžeme ďalej spracovávať. Výstup bude teda obrázok rozmerov porovnávaných obrázkov vo formáte PBM, kde biele miesto bude značiť rovnaké pixely a čierne body budú udávať rozdielne pixely dvoch porovnávaných obrázkov. Takýto výstupný formát nie je práve najúspornejší, ale je prenositeľný a dokážeme ho zobraziť aj v konzole.

Číselný výstup

Ak ale nepotrebujeme vedieť, ktoré pixely sú rozdielne ale zaujíma nás len v akej miere sa dva obrázky podobajú, môžeme túto podobnosť vyjadriť číslom – mierou na koľko percent sú si dva obrázky podobné. Táto voľba bude vhodná najmä pri strojovom spracovávaní veľkého množstva obrázkov.

Transformácie

Pomerne často sú obázky rôzne transformované – otočené, zrkadlené, zmenšené alebo zväčšené. Program `mdiff` by mal dva rovnaké obrázky, z ktorých jeden je nejak transformovaný, prehlásiť za zhodné a popísať, aké transformácie boli použité. `Mdiff` bude prijímať rôzne voľby, ktoré povolujú skúšať transformácie na obrázkoch za cieľom dosiahnutia čo najlepšej podobnosti. Pri otočení vypočítame mieru podobnosti pre všetky štyri možné otočenia a vyberieme najlepšiu. Ak povolíme zrkadlenie, skúsime obrázok zrkadliť zľava do prava a z hora dole, pre každý prípad vypočítame mieru podobnosti a nakoniec vyberieme najlepšiu z nich. Ak povolíme obe voľby, otočenie aj zrkadlenie zároveň, program musí vypočítať mieru podobnosti pre všetky možné spôsoby transformácií – čiže 4 otočenia $\times$ 2 zrkadlenia $= 8$ porovnávaní. Týmito voľbami sa program výrazne spomalí. Poslednou transformáciou je zmenšenie, respektíve zväčšenie. Na zistenie, či je jedne obrázok rovnaký ako druhý, ktorý je zmenšený (zväčšený), zavedieme ďalšiu voľbu, ktorá nám transformuje obrázky do nami zadaného rozlíšenia a porovná ich. Túto voľbu môžeme použiť ak chceme porovnávanie výrazne urýchliť. Program bude mať voľbu, ktorá zmenší obrázky do malého rozlíšenia, napríklad 64×64 pixelov a tak ich porovná.

Možné problémy

Niektoré grafické formátu používajú stratovú kompresiu. Obrázok je síce na pohľad rovnaký, ale pri detailnom skúmaní zisťujeme, že farba niektorých pixelov je odlišná. Ak porovnávame dva rovnaké obrázky používajúce stratovú kompresiu s rôznym komprimačným pomerom, nebudú na 100% rovnaké. Čiastočné riešenie tohto problému je zavedenie toleranice. Voľbou môžeme zadať toleranciu t v rozsahu 0-1, potom dva pixely x_{ij} a y_{ij} budú označené za zhodné, ak $(1-t)y_{ij} \leq x_{ij} \leq (1+t)y_{ij}$.

Ďalší problém, ktorý musíme riešiť, je poradie aplikácií transformácií. Ak porovnávame dva obrázky uložené v bezstratovom formáte a najskôr ich zmenšíme, následne otočíme a porovnáme, dostaneme inú mieru podobnosti ako keby sme najskôr otočili a až potom zmenšili a porovnali. Toto je spôsobené interpoláciou pixelov pri zmenšovaní. Rotácie a zrkadlenie obrázku nespôsobujú interpolačné zmeny, preto na obrázok aplikujeme tieto transformácie. Až nakoniec obrázok zmenšíme. Takto dosiahneme toho, že ak budú dva obrázky rovnaké a jeden z nich pootočený (zrkadlený), náš program vhodnými transformáciami a porovnaním potvrdí zhodnosť obrázkov.

Kapitola 6

Implemetácia a testovanie

6.1 Implementácia

Implementáciu celého programu sme rozdelili do modulov. Aby boli zdrojové kódy dobre šíriteľné, bola pri komentároch a ostatných názvoch použitá angličtina. V prílohe D nájdeme nápovedu k programu `mdiff`. Následne si popíšeme jednotlivé moduly tak ako sme si ich navrhli v predchádzajúcej kapitole.

6.1.1 Modul mdiff.py

Hlavným modulom je `mdiff.py`. Hlavnou úlohou tohto modulu je spracovávať argumenty príkazového riadku a volať príslušné funkcie. Tento modul bude obsahovať funkciu `main`, ktorá bude ako prvá spustená po štarte programu. V tejto funkcii sa načítajú argumenty príkazového riadku, následne sa vyšetria a zavolá sa požadovaná funkcia. Na vyšetrenie argumentov bola použitá knižnica `optparse`. Ak nebudú zadané argumenty príkazového riadku, pomocou knižnice `mimetypes` sa program sám pokúsi uhádnuť typ súborov.

6.1.2 Modul misc.py

Po spracovaní argumentov budeme potrebovať oba súbory otvoriť. Funkciu na bezpečné otvorenie súboru, načítanie jeho obsahu do pamäti a zatvorenie súboru umiestnime do zvláštneho modulu `misc.py`. Takto bude existovať len jedna funkcia, ktorú budeme používať aj v ostatných moduloch a vyhneme sa duplicite kódu.

6.1.3 Modul difflib2.py

Jeden z najdôležitejších modulov je upravená knižnica `difflib` – `difflib2`. Táto knižnica obsahuje triedu `SequenceMatcher`, v ktorej je implementovaný algoritmus vychádzajúci z algoritmu autorov Ratcliffa a Obershelpa [18]. Základný algoritmus má v najhoršom prípade kubickú zložitosť a kvadratickú v bežnom prípade. Algoritmus implementovaný v triede `SequenceMatcher` má v najhoršom prípade kvadratickú zložitosť, zložitosť v bežnom prípade závisí od počtu spoločných elementov sekvencií, v najlepšom prípade má algoritmus lineárnu zložitosť.

Konštruktor triedy prijíma dve sekvencie (`a` a `b`), v našom prípade zoznamy riadkov. Aby sme dosiahli ignorovanie veľkosti písmen alebo bielych znakov, upravili sme konštruktor triedy. Nový konštruktor prijíma tretí argument (`prepare`) – funkciu, ktorá prijíma

sekvenciu (`a` alebo `b`) a vracia tú istú sekvenciu, v ktorej sú ale jednotlivé položky pozmenené (napríklad vymazanie prebytočných medzier v riadkoch). Takýmto spôsobom priamo pri konštrukcii nového objektu vytvoríme pozmenené kópie oboch sekvencií, ktoré sa budú porovnávať. Funkciu, ktorá upravuje sekvencie si popíšeme neskôr.

Ďalej sme rozšírili výstupné formáty knižnice difflib o normálny formát popísaný na strane 13.

6.1.4 Modul textdiff.py

Modul `textdiff.py` ponúka rozhranie na porovnávanie textov. Obsahuje triedu `TextDiff`, ktorá obsahuje metódy na porovnávanie textov a zobrazenie výstupu v jednom z uvedených formátov. Ďalšou metódou je metóda `prepare_lines(self,lines)`, ktorá sa predáva ako posledný parameter pri konštrukcii objektu triedy `SequenceMatcher`. Táto metóda podľa volieb pri spustení programu `mdiff` upraví riadky porovnávaných textov (ignorovanie bielych znakov, veľkosti písmen alebo časti riadku definovaného regulárnym výrazom).

6.1.5 Modul latexdiff.py

Modul, ktorý pomocou regulárnych výrazov odstráni značky z latexových súborov a za použitia modulu `textdiff` tieto vyextrahované texty porovná a zobrazí.

6.1.6 Modul configdiff.py

Ako sme popísali v analýze, načítame oba súbory po riadkoch (vynecháme komentáre), prvý do zoznamu, druhý do slovníku (asociatívne pole). Priechodom cez zoznam riadkov a porovnávaním týchto riadkov so slovníkom rozdelíme riadky na tri skupiny, ktoré nakoniec vypíšeme.

6.1.7 Modul fileinfo.py

Pomocou štandardných knižníc získame informácie o oboch súboroch. Následne tieto informácie porovnáme a zobrazíme v formáte ndiff.

6.1.8 Modul oodiff.py

Z oboch ODF súborov – archívov vyextrahujeme súbory content.xml, ktoré obsahujú text práce. Z tohto súboru za pomoci regulárnych výrazov vyextrahujeme samotný text. Tieto texty pomocou modulu `textdiff.py` porovnáme a zobrazíme v požadovanom výstupnom formáte.

6.1.9 Modul imagediff.py

V čase tvorenia práce bola použitá na implementáciu hlavného programu čerstvá verzia jazyku Python 3.0. Knižnice na prácu s obrázkami však za tak krátky čas neboli vydané pre Python 3.0, preto bola použitá knižnica Python Imaging Library (PIL) pre Python 2.5. Z tohoto dôvodu sme zvolili dočasné riešenie a tento modul tvorí samostatný program písaný v jazyku Python 2.5 a nie je súčasťou programu `mdiff`. Keď sa knižnica PIL aktualizuje na verziu pre Python 3.0, bude sa musieť tento modul aktualizovať a pripojiť k programu `mdiff`.

Modul `imagediff` je spustiteľný a sám spracuváva argumenty príkazového riadku. Jeho použitie je rovnaké ako pri programe `mdiff`.

Tento modul obsahuje rovnomennú triedu, ktorá poskytuje metódy na porovnávanie obrázkov. Hlavná metóda je `ratio(image1,image2,toleration=0)`. Táto metóda prechádza po pixeloch oboch obrázkov a vypočíta percentuálnu podobnosť oboch obrázkov. Berie aj argument toleration – tolerancia (význam popísaný v analýze).

Malou optimalizáciou tejto funkcie je vytvorenie dvoch cyklov, ktoré oba prechádzajú cez pixely a porovnávajú ich. Jedna verzia porovnáva pixely na zhodnosť, druhá pracuje aj s toleranciou. Takto sme ušetrili niekoľko porovnaní pixelov vnútri vnoreného cyklu, čo výrazne urýchlilo porovnávanie pri nulovej tolerancii.

Ďalšou metódou je `best_ratio(image1,image2)`, ktorá, ak sme zadali volby otočenie a zrkadlenie obrázku, sa pokúša nájsť najlepšiu zhodu dvoch obrázkov za pomoci povolených transformácií.

Posledná metóda `pbm_otput(image1,image2)` pomocou predchádzajúcich metód vytvorí obrázok vo formáte PBM, v ktorom sú zakreslené odlišné pixely.

6.2 Testovanie

6.2.1 Porovnanie programov `mdiff` a `diff`

Keďže program `mdiff` má nahradiť program `diff`, skúsime si tieto dva programy porovnať.

Najskôr budeme porovnávať dva textové súbory veľkosti 1 026 riadkov (98,8 KB).

Typ testu	`GNU diff`	`mdiff`
Porovnávanie rovnakých súborov	0,007 s	0,215 s
Zmena v jednom znaku	0,008 s	0,208 s
Viac zmien v oboch súboroch	0,010 s	0,207 s
Dva rozdielne texty	0,011 s	0,179 s

Tabulka 6.1: Test 1 programov `GNU diff` a `mdiff`

Rovnaké testy prevedieme na dvoch ďalších textových súboroch, tentokrát ale omnoho väčších – 91 522 riadkov (8,8 MB).

Typ testu	`GNU diff`	`mdiff`
Porovnávanie rovnakých súborov	0,117 s	7,076 s
Zmena v jednom znaku	0,17 8s	7,217 s
Viac zmien v oboch súboroch	0,226 s	7,081 s
Dva rozdielne texty	0,461 s	10,881 s

Tabulka 6.2: Test 2 programov `GNU diff` a `mdiff`

Vyskúšame ešte rýchlosť programov pri porovnávaní textov z prvého testu, ale zapneme postupne jednotlivé voľby ako ignorovanie veľkosti písmen alebo počtu bielych znakov.

Typ testu	`GNU diff`	`mdiff`
Klasické porovnanie	0,010 s	0,205 s
Ignorovanie bielych znakov	0,010 s	0,253 s
Ignorovanie veľkosti písmen	0,011 s	0,237 s
Ignorovanie aj bielych znakov aj veľkosti písmen	0,012 s	0,261 s

Tabulka 6.3: Test 3 programov `GNU diff` a `mdiff`

6.2.2 Test rýchlosti pri porovnávaní rôznych typov dokumentov

V ďalšej tabuľke si ukážeme rýchlosti porovnávania jednotlivých dokumentov. Pri textoch sa budeme snažiť dodržať dĺžku 1 000 riadkov.

Typ testu	`mdiff`
Porovnávanie latexových súborov	0,575 s
Porovnávanie ODT súborov	1,460 s
Porovnávanie konfiguračných súborov	0,254 s
Porovnávanie informácií o súboroch	0,137 s

Tabulka 6.4: Test 4 porovnávanie rôznych typov dokumentov

6.2.3 Test porovnávania obrázkov

Na záver si uvedieme testy posledného modulu – `imagediff`. Na ľavom obrázku 6.1a je hore vľavo originálny obrázok, vpravo je ten istý so zmenami (oči), dole vľavo pozmenený a otočený obrázok a posledný obrázok je rozdiel originálu voči pozmenenému obrázku.

Pre dvojicu obrázkov 6.1a a 6.1b nám `imagediff` vypočítal zhodnosť na 99,899 %. Ak porovnáme originálny obrázok 6.1a s 6.1c, výsledná podobnosť je 0.193 %. Ak povolíme rotácie a zrkadlenie obrázky sa budú zhodovať na 99,899 %. Na poslednom obrázku 6.1d je rozdiel originálu a pozmeneného obrázku vo formáte PBM.

Predchádzajúce obrázky boli v BMP formáte, ktorý je bezstratový. Ak budeme porovnávať obrázky uložené v niektorom zo stratových formátov, výsledky budú úplne odlišné. Napríklad ak porovnáme obrázky C.1a a C.1b uložené v stratovom formáte JPEG, budú si podobné len na 84.534 %. Keď ale povolíme toleranicu 0,2, výsledná podobnosť stúpne na 96.947 %. Na obrázku C.1c je výstup po porovnaní s nulovou toleranciou, na vedľajšom obrázku je povolená tolerancia 0,2. Ako je vidieť, porovnávanie stratových formátov je značne nepresné.

Na koniec si uvedieme časové údaje v tabuľke C.1. Význam jednotlivých volieb nájdeme v prílohe D.

Obrázek 6.1: a) originálny obrázok, b) pozmenený obrázok, c) pozmenený, otočený a zrkadlený obrázok, d) rozdiel obrázkov

Voľby	1. obr	2. obr	čas	ratio	poznámky
`-r`	6.1a	6.1b	1,329 s	99,899 %	
`-r -f`	6.1a	6.1b	0.146 s	99.9029 %	
`-t`	6.1a	6.1b	2,830 s	99,899 %	
`-r -R -F`	6.1a	6.1c	9,522 s	99.899 %	Flip: Flip left right, Angle: 90
`-r -f -R -F`	6.1a	6.1c	0.264 s	99.902 %	Flip: Flip left right, Angle: 90
`-r`	C.1a	C.1b	0.503 s	84.534 %	
`-r -T 0.1`	C.1a	C.1b	0.622 s	95.938 %	
`-r -T 0.2`	C.1a	C.1b	0.605 s	96.947 %	

Tabulka 6.5: Časy a výsledky porovnávania obrázkov

Kapitola 7

Záver

Cieľom tejto práce bolo vyvinúť program – `mdiff`, ktorý bude porovnávať rôzne typy dokumentov. Tento cieľ sa mi podaril splniť a vytvoril som plne funkčnú aplikáciu.

Táto aplikácia doplní prázdnu dieru vedľa programu `diff`, pretože doteraz neexistoval jeden nástroj porovnávajúci viac rozmanitých typov súborov. Užívateľ tak nemusí viac hľadať rôzne nástroje na porovnávanie rôznych súborov ako tomu bolo doteraz. V kancelárskom prostredí bude veľmi cenené porovnávanie ODT súborov. Pri práci s grafikou a obrázkami zas môžeme využiť porovnávanie obrázkov, ktoré má množstvo volieb a dokáže nájsť zhodu, aj keď je jeden z obrázkov otočený zrkadlený alebo zmenšený. To nám pomôže nájsť napríklad duplicitné fotky v albume. Pri písaní prácí v latexe dokážeme s `mdiff`om čitateľne zobraziť zmeny medzi rôznymi verziami. Za zmienku stojí aj porovnávanie konfiguračných súborov. Porovnávanie textov s veľkým množstvom volieb je samozrejmosťou.

Program `mdiff` je tvorený ako modulárna aplikácia a jednotlivé moduly môžu byť použité ako knižnice v iných programoch.

Pri testovaní programu sme dosiahli veľmi dobré výsledky. Pri porovnávaní textu bol síce `mdiff` oveľa pomalší než `diff`, ale výsledný čas pri texte bežnej dĺžky bol pod pol sekundy. Napriek tomu pri obrovských súboroch o cca 90 000 riadkoch trvalo porovnávanie 7 s. Tento čas nie je zlý, ale je rádovo vyšší než `diff`. Spôsobené je to z časti tým, že `mdiff` beží v interpretovanom jazyku a z časti tým, že `diff` je rokmi preverená a vysoko optimalizovaná aplikácia. Porovnávanie obrázku o veľkosti 512 × 512 pixelov trvalo 2,8 s. Pri povolení otáčania a zrkadlenia sme sa dostali až k 9,5 s. Tento čas je už pomerne vysoký, ale keď sme použili rýchle porovnávanie, dosiahli sme čas 0,14 s s odchýlkou 0,003 %, čo je vynikajúci výsledok.

Počas vývoja tejto aplikácie som narazil na niekoľko rôznych problémov. Jedným z problémov bolo upravenie štandardnej knižnice difflib jazyku Python. Po detailnom naštudovaní celého zdrojového kódu knižnice som doimplementoval funcionalitu, ktorá umožňuje porovnávanie dvoch sekvencií so zanedbaním niektorých aspektov a pridal som Normálny formátu výstupu 3.1.3. Dúfam, že mojím rozšírením knižnice difflib prispejem k vývoju jazyka Python a že táto knižnica bude v ďalšej verzii jazyka.

Nakoľko rozsah tejto práce nepokrýva implementáciu porovnávania všetkých bežne používaných typov dokumentov, je možné ďalšiu funcionalitu doplniť. Vítaným rozšírením je podpora porovnávania audia a videa alebo spojenie s programom `xmldiff`, poprípade grafické užívateľské prostredie.

Niektoré texty tejto práce budú použité na rozšírenie slovenskej wikipédie.

Literatura

[1] pbm - portable bitmap file format [online]. `http://netpbm.sourceforge.net/doc/pbm.html`, [cit. 2009-05-12].

[2] Cormen, T. H.; et al.: *Introduction to algorithms*. Massachusetts: MIT Press, druhé vydání, 2001, 350–355 s., iSBN 0-262-03293-7.

[3] Hamming, R. W.: Error detecting and error correcting codes. *Bell System Technical Journal*, ročník 29, č. 2, 1950: s. 147–160.

[4] Hunt, J. W.; McIlroy, M. D.: An Algorithm for Differential File Comparison. Technická Zpráva CSTR 41, Bell Laboratories, Murray Hill, NJ, 1976.

[5] Kolektív autorov: diff [online]. `http://en.wikipedia.org/wiki/Diffutils`, [cit. 2009-04-21].

[6] Kolektív autorov: Levenshtein distance [online]. `http://en.wikipedia.org/wiki/Levenshtein_distance`, [cit. 2009-05-09].

[7] Kolektív autorov: BMP [online]. `http://cs.wikipedia.org/wiki/BMP`, [cit. 2009-05-12].

[8] Kolektív autorov: Configuration file [online]. `http://en.wikipedia.org/wiki/Config_file`, [cit. 2009-05-12].

[9] Kolektív autorov: Geany [online]. `http://www.geany.org/`, [cit. 2009-05-12].

[10] Kolektív autorov: Longest common subsequence problem [online]. `http://en.wikipedia.org/wiki/Longest_common_subsequence_problem`, [cit. 2009-05-12].

[11] Kolektív autorov: Open Document Format for Office Applications (OpenDocument) v1.1 [online]. `http://docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1-html/OpenDocument-v1.1.html`, [cit. 2009-05-12].

[12] Kolektív autorov: OpenDocument [online]. `http://cs.wikipedia.org/wiki/OpenDocument`, [cit. 2009-05-12].

[13] Kolektív autorov: Portable Network Graphics [online]. `http://cs.wikipedia.org/wiki/Portable_Network_Graphics`, [cit. 2009-05-12].

[14] Kolektív autorov: Subversion [online]. `http://subversion.tigris.org/`, [cit. 2009-05-12].

[15] MacKenzie, D.; Eggert, P.; Stallman, R.: *Comparing and Merging Files with GNU Diff and Patch*. Bristol: Network Theory, 1997, iSBN 0-9541617-5-0.

[16] Miller, W.; Myers, E. W.: A File Comparison Program. *Software–Practice and Experience*, ročník 15, č. 11, 1985: s. 1025–1040.

[17] Myers, E. W.: An O(ND) Difference Algorithm and its Variations. *Algorithmica*, ročník 1, č. 2, 1986: s. 251–266.

[18] Ratcliff, J.; Metzener, D.: Pattern Matching: The Gestalt Approach. *Dr. Dobb's Journal*, 1988: str. 46.

[19] Ukkonen, E.: Algorithms for Approximate String Matching. *Information and Control*, ročník 64, 1985: s. 100–118.

Seznam obrázků

Seznam tabulek

Dodatek A

Príklady formátov výstupu

Vzorové texty a niektoré uvedené formáty výstupu sú prebrané z [15].

A.1 Vzorové texty

Súbor lao:

```
The Way that can be told of is not the eternal Way;
The name that can be named is not the eternal name.
The Nameless is the origin of Heaven and Earth;
The Named is the mother of all things.
Therefore let there always be non-being,
  so we may see their subtlety,
And let there always be being,
  so we may see their outcome.
The two are the same,
But after they are produced,
  they have different names.
```

Súbor tzu:

```
The Nameless is the origin of Heaven and Earth;
The named is the mother of all things.

Therefore let there always be non-being,
  so we may see their subtlety,
And let there always be being,
  so we may see their outcome.
The two are the same,
But after they are produced,
  they have different names.
They both may be called deep and profound.
Deeper and more profound,
The door of all subtleties!
```

V týchto príkladoch sú niektoré riadky z prvého súboru zmazané, pridané do súboru lao alebo zmenené v oboch súboroch navzájom.

A.2 Normálny formát výstupu

Príklad normálneho formátu výstupu:

```
1,2d0
< The Way that can be told of is not the eternal Way;
< The name that can be named is not the eternal name.
4c2,3
< The Named is the mother of all things.
---
> The named is the mother of all things.
>
11a11,13
> They both may be called deep and profound.
> Deeper and more profound,
> The door of all subtleties!
```

A.3 Kontextový formát výstupu

Príklad kontextového formátu výstupu:

```
*** lao 2002-02-21 23:30:39.942229878 -0800
--- tzu 2002-02-21 23:30:50.442260588 -0800
***************
*** 1,7 ****
- The Way that can be told of is not the eternal Way;
- The name that can be named is not the eternal name.
  The Nameless is the origin of Heaven and Earth;
! The Named is the mother of all things.
  Therefore let there always be non-being,
    so we may see their subtlety,
  And let there always be being,
--- 1,6 ----
  The Nameless is the origin of Heaven and Earth;
! The named is the mother of all things.
!
  Therefore let there always be non-being,
    so we may see their subtlety,
  And let there always be being,
***************
*** 9,11 ****
--- 8,13 ----
  The two are the same,
  But after they are produced,
    they have different names.
+ They both may be called deep and profound.
+ Deeper and more profound,
+ The door of all subtleties!
```

A.4 Unifikovaný formát výstupu

Príklad unifikovaného formátu výstupu:

```
--- lao 2002-02-21 23:30:39.942229878 -0800
+++ tzu 2002-02-21 23:30:50.442260588 -0800
@@ -1,7 +1,6 @@
-The Way that can be told of is not the eternal Way;
-The name that can be named is not the eternal name.
 The Nameless is the origin of Heaven and Earth;
-The Named is the mother of all things.
+The named is the mother of all things.
+
 Therefore let there always be non-being,
   so we may see their subtlety,
 And let there always be being,
@@ -9,3 +8,6 @@
 The two are the same,
 But after they are produced,
   they have different names.
+They both may be called deep and profound.
+Deeper and more profound,
+The door of all subtleties!
```

A.5 Formátu výstupu vedľa seba

Príklad formátu výstupu vedľa seba:

```
The Way that can be told of is n   <
The name that can be named is no   <
The Nameless is the origin of He       The Nameless is the origin of He
The Named is the mother of all t   |    The named is the mother of all t
                                   >
Therefore let there always be no       Therefore let there always be no
  so we may see their subtlety,          so we may see their subtlety,
And let there always be being,         And let there always be being,
  so we may see their outcome.           so we may see their outcome.
The two are the same,                  The two are the same,
But after they are produced,           But after they are produced,
  they have different names.             they have different names.
                                   >    They both may be called deep and
                                   >    Deeper and more profound,
                                   >    The door of all subtleties!
```

Aby sa nám výsledok zmestil na stránku, museli sme obmedziť šírku textu na 72 znakov.

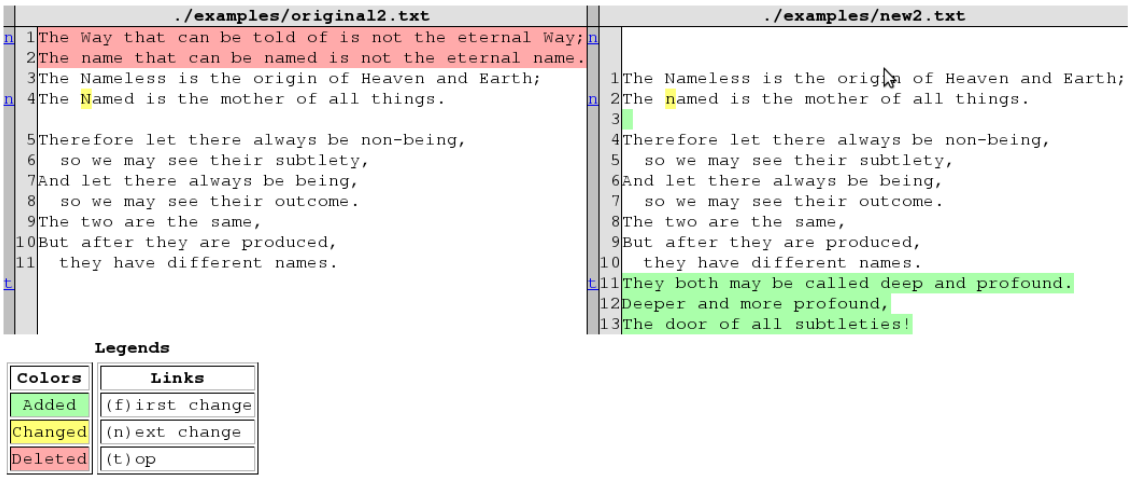
A.6 Formát výstupu ndiff

Riadky ktoré sú rovnaké začínajú dvoma medzerami, riadky ktoré sú zmazane z prvého súboru začínajú znakom `-` a medzerou, vložené riadky do druhého súboru začínajú znakom `+`. Špeciálne riadky začínajúce znakom `?` majú informatívny charakter a sú vložené pod riadky, ktoré sa líšia iba v určitej miere. V tomto pomocnom riadku ukazujú znaky `^` na znaky predchádzajúceho riadku, ktoré sú odlišné.

Príklad formátu výstupu ndiff:

```
- The Way that can be told of is not the eternal Way;
- The name that can be named is not the eternal name.
  The Nameless is the origin of Heaven and Earth;
- The Named is the mother of all things.
?     ^
+ The named is the mother of all things.
?     ^
+
  Therefore let there always be non-being,
    so we may see their subtlety,
  And let there always be being,
    so we may see their outcome.
  The two are the same,
  But after they are produced,
    they have different names.
+ They both may be called deep and profound.
+ Deeper and more profound,
+ The door of all subtleties!
```

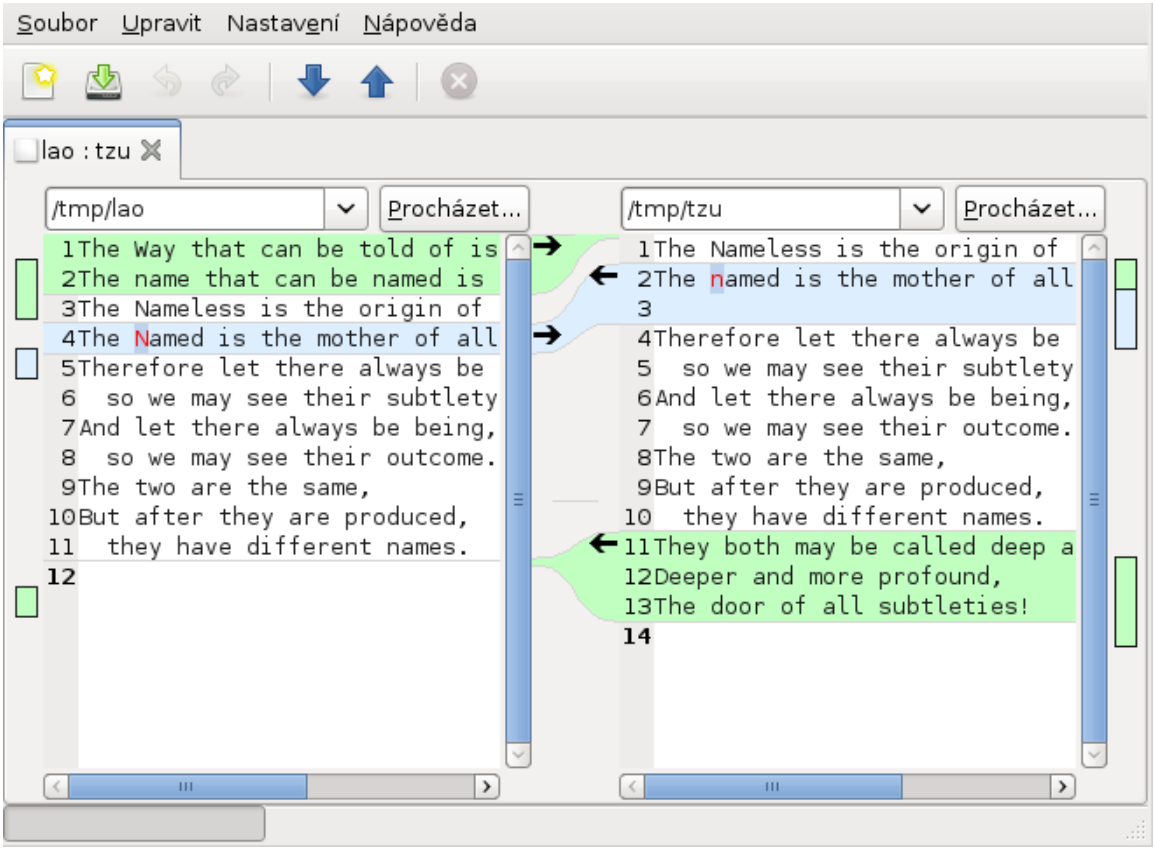
A.7 HTML výstup



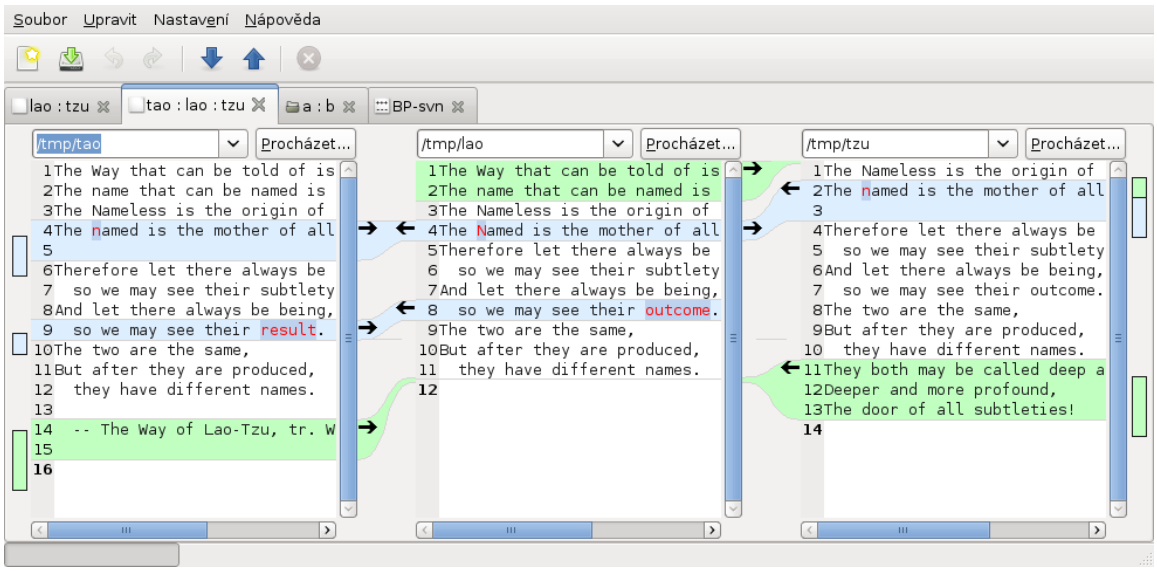
Obrázek A.1: Výstup vo formáte html

Dodatek B

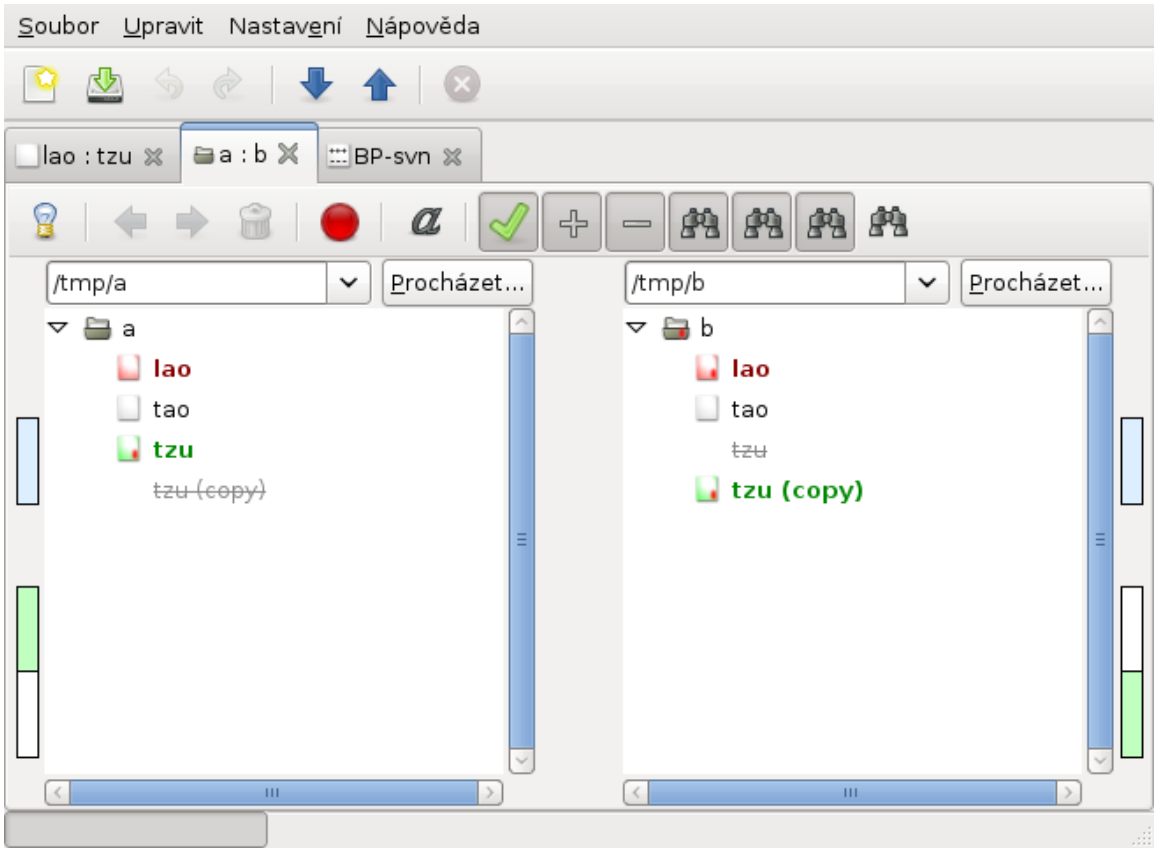
Ukážky programu meld



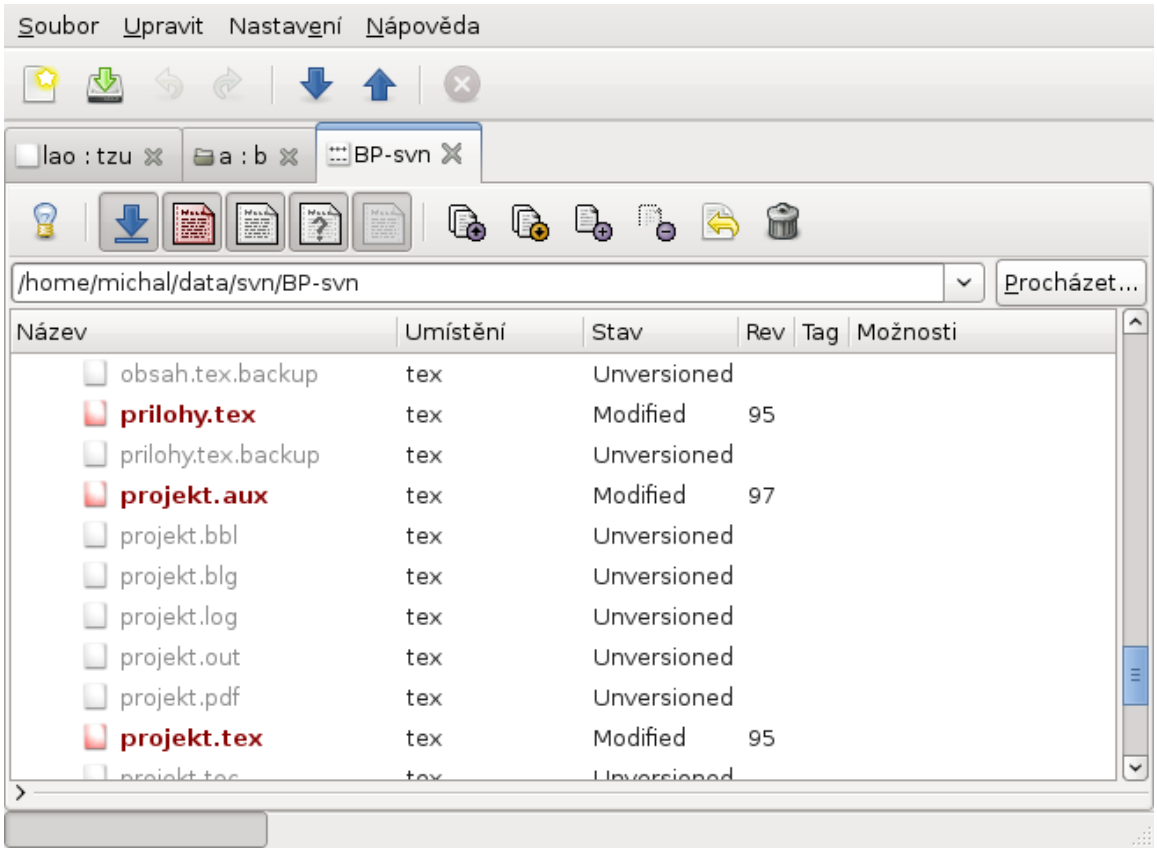
Obrázek B.1: Porovnávanie dvoch súborov



Obrázek B.2: Trojcestné porovnávanie



Obrázek B.3: Porovnávanie zložiek



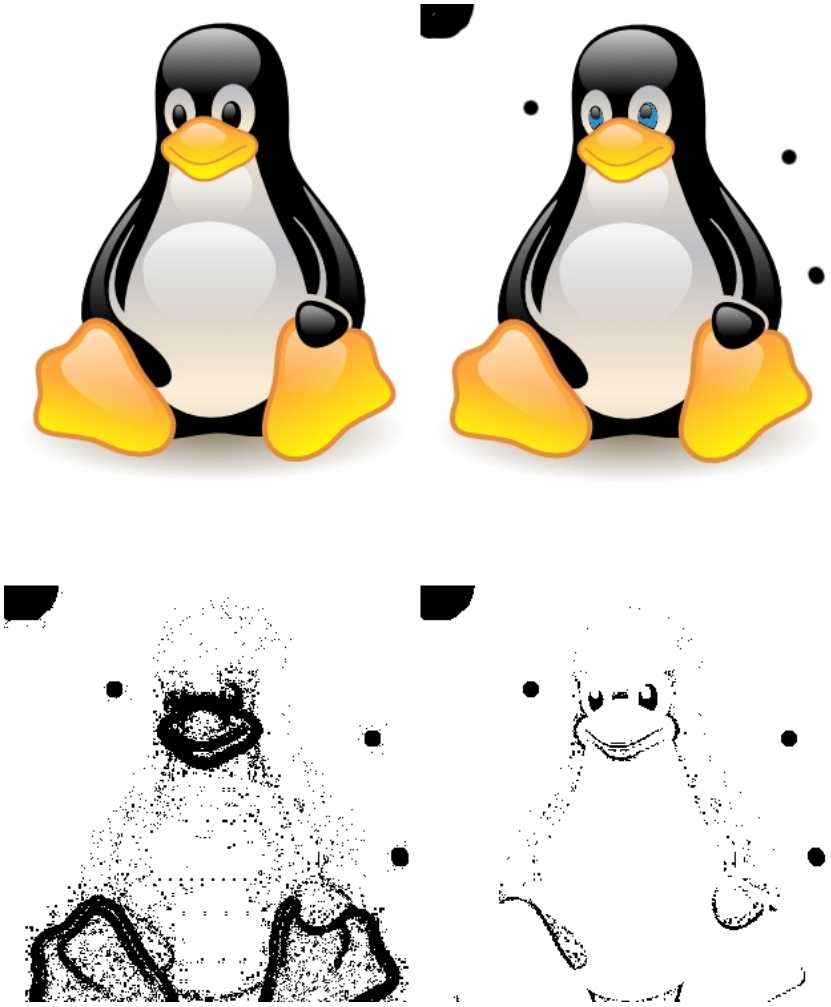
Obrázek B.4: Prechádzanie svn súborov

Dodatek C

Test porovnávania obrázkov

Voľby	1. obr	2. obr	čas	ratio	poznámky
`-r`	6.1a	6.1b	1,329 s	99,899 %	
`-r -S 128 128`	6.1a	6.1b	0.180 s	99.902 %	
`-r -f`	6.1a	6.1b	0.146 s	99.9029 %	
`-t`	6.1a	6.1b	2,830 s	99,899 %	
`-r`	6.1a	6.1c	1,362 s	0.193 %	
`-r -R`	6.1a	6.1c	5,049 s	0.199 %	Angle: 180
`-r -F`	6.1a	6.1c	4,222 s	0.193 %	Angle: 180
`-r -R -F`	6.1a	6.1c	9,522 s	99.899 %	Flip: Flip left right, Angle: 90
`-r -f -R -F`	6.1a	6.1c	0.264 s	99.902 %	Flip: Flip left right, Angle: 90
`-r -R -F`	6.1b	6.1c	9,528 s	100.0 %	Flip: Flip left right, Angle: 90
`-r`	C.1a	C.1b	0.503 s	84.534 %	
`-r -T 0.1`	C.1a	C.1b	0.622 s	95.938 %	
`-r -T 0.2`	C.1a	C.1b	0.605 s	96.947 %	
`-r -T 0.3`	C.1a	C.1b	0.632 s	97.345 %	

Tabulka C.1: Časy a výsledky porovnávania obrázkov



Obrázek C.1: a) originálny obrázok, b) pozmenený obrázok, c) rozdiel obrázkov, d) rozdiel obrázkov pri tolerancii 20 %

Dodatek D

Nápoveda k programu `mdiff`

D.1 Nápoveda k `mdiff`

```
michal@b07-606a:~/data/svn/BP-svn/mdiff$ ./mdiff -h
Usage: mdiff [options] from_file to_file

Multiple Document Type Diff, compares various types of files.

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit

  Text differencing tools:
    -d, --standard-diff
                        Use of standard GNU diff output format.
    -n, --normal-diff   Use of ndiff output format.
    -c, --context-diff  Use of the context output format.
    -C lines, --context=lines
                        Set, number of context lines. Default is three.
                        Use of this option with option -c
    -u, --unified-diff  Use of the unified output format.
    -v, --html-diff     Produces HTML side by side diff (can use -c and -C
     in conjunction)

  Other files differencing tools:
    -f, --file-info     Compares file informations. Shows only different
                        attributes
    -F                  Compares file informations. Shows all attributes
    -o, --open-office   Compares Open Office Text files.
    -O characters, --line-width=characters
                        Width of lines in comparation Open Office files.
    -l, --latex         Compares latex files.
    -k, --config-files  Compares config files.

  Advanced options for text differencing tools:
    -i, --ignore-case   Ignores case differences in file contents.
```

```
    -b, --ignore-space-change
                        Ignores changes in the amount of white space.
    -w, --ignore-all-space
                        Ignores all white spaces, but blank lines are
                        deleted and then texts are compared. Indices
                         doesn't have to be correct. It is recommended
                         to use the ndiff output format.
    -B, --ignore-blank-lines
                        Ignores changes whose lines are all blank.
                         Blank lines are deleted and then texts are
                         compared. Indices doesn't have to be correct. It
                         is recommended to use the ndiff output format.
    -I RE, --ignore-matching-lines=RE
                        Ignores changes whose lines all match RE. Matching
                        lines are deleted and then compared. Indices
                        doesn't have to be correct. It is recommended to
                         use the ndiff output format.
    -J RE, --ignore-matching-words=RE
                        Ignores changes whose words all match RE.
```

D.2 Nápoveda k imagediff

```
michal@b07-606a:~/data/svn/BP-svn/mdiff$ ./imagediff -h
Usage: imagediff [options] from_file to_file

Image Diff compares images and shows their differences.

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -t, --text-otput      Shows text output in PBM format.
  -r, --ratio           Shows image similarity percentage.
  -f, --fast            Resizes images to 64x64 pixels and faster shows the
                        ratio.
  -R, --enable-rotation
                        Rotates images to find the best ratio.
  -F, --enable-flip     Flips images to find the best ratio.
  -T toleration, --toleration=toleration
                        Toleration (0-1) between each pixel colors.
  -S x y, --size=x y    Resizes images for faster comparation.
```