VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

SKRIPTY PRO HROMADNOU ÚPRAVU FONTŮ V PDF DOKUMENTECH

SCRIPTS FOR AUTOMATED EDITING OF FONTS IN PDF FILES

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Jakub Gmitter

AUTHOR

VEDOUCÍ PRÁCE — Ing. Pavel Hanák, Ph.D.

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Jakub Gmitter ***ID:*** 220814

Ročník: 2 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Skripty pro hromadnou úpravu fontů v PDF dokumentech

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s vnitřní strukturou PDF dokumentů, zejména s typy fontů, jejich kódováním a ToUnicode CMap tabulkami. Následně vytvořte skripty, které budou schopny detekovat chybějící ToUnicode mapování a u vybraných fontů jej doplnit. K tomu využijte buď uživatelem vytvořené korekční tabulky nebo tabulky extrahované ze zabudovaných fontů. Skripty musejí umět automaticky či polo-automaticky zpracovat větší množství podobných PDF dokumentů a uživatele upozornit na chyby, které při tom mohou vzniknout. Jejich funkci ověřte nejméně na 20 vzorcích, které vám poskytne vedoucí. Korekční tabulky a konfigurační soubory musejí být koncipovány tak, aby je bylo možné snadno adaptovat na jiné dokumenty. Pokud to bude možné, tak při tvorbě využijte pouze open-source softwarové balíky jako PDFminer.six, PikePDF, MuPDF nebo FontForge. Skripty opatřete podrobnými komentáři funkce a návodem k použití.

DOPORUČENÁ LITERATURA:

[1] Adobe Systems Incorporated, Adobe CMap and CIDFont Files Specification, 1993. Dostupné online https://adobe-type-tools.github.io/font-tech-notes/pdfs/5014.CIDFont_Spec.pdf

[2] Summerfield, M. Python 3. Computer Press, 2021, ISBN 978-80-251-5030-6

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 21.5.2024

Vedoucí práce: Ing. Pavel Hanák, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Diplomová práca sa venuje problematike kódovania fontov v PDF dokumentoch. Správne kódovanie fontov je potrebné pre vyhľadávanie v dokumente a kopírovanie textu z dokumentu. Práca obsahuje popis vnútornej štruktúry PDF dokumentov, reprezentáciu strán, typy fontov a ich kódovanie a dôležité objekty potrebné pre správnu reprezentáciu fontov. Znalosti z týchto oblastí sú kľúčové pre vývoj skriptov na opravu kódovania fontov. V rámci diplomovej práce boli vytvorené dva skripty v jazyku Python. Prvý zo skriptov overuje integritu opravovaných PDF súborov pomocou hešov SHA-256 vypočítaných z ich obsahu. Druhý skript opravuje poškodené kódovania fontov v dokumentoch. Potrebné informácie pre funkčnosť oboch skriptov boli uložené do zodpovedajúcich JSON štruktúr. Opravný skript sa zameriava na PostSciptové fonty typu 1. Kľúčovým prvkom opravného skriptu je generovanie objektu ToUnicode, ktorý v rámci fontu správne mapuje glyfy na Unicode kódy. Skript bol testovaný na približne 200 elektronických vydaniach českého časopisu, ktoré boli poskytnuté ako vzorové údaje. Zo vzorových súborov boli vybrané tie, ktoré mali kompletne poškodené kódovania fontov. Ostatné vzorové časopisy mali poškodené iba kódovanie znakov s diakritickými znamienkami. Tieto časopisy boli analyzované, ale skript ich nedokáže opraviť. Komentované zdrojové kódy jazyka Python, skompilované spustiteľné súbory systému Windows a používateľská príručka sú k dispozícii v elektronickej prílohe a v autorovom GitHub repoziráti.

KĽÚČOVÉ SLOVÁ

Font, JSON, PDF, Python, Skript, Unicode

ABSTRACT

Master's thesis deals with the issue of font encoding in PDF documents. Proper font encoding is necessary for searching and copying text from such documents. Thesis includes a description of the internal structure of PDF documents, page representation, font types and their encoding, and important objects needed for proper font representation. Understanding of these areas was necessary for development of scripts that are able to repair incorrect font encoding. Two Python scripts were developed as part of the thesis. The first one verifies the integrity of repaired PDF files using SHA-256 hashes computed from their contents. The second script repairs corrupted font encodings in the documents. The necessary information for the functionality of both scripts has been stored in the corresponding JSON structures. The repair script targets PostScipt fonts of type 1. Core function of the repair script is the generation of a ToUnicode object that correctly maps glyphs to Unicode codes within the font. The script has been tested on approximately 200 electronic issues of a Czech magazine that have been provided as sample data. From these sample files, only those that had completely corrupted font encodings were chosen for further work. Other sample magazines only had corrupt encoding of characters with diacritical marks. These magazines were analyzed, but the script is unable to repair them. Commented Python source code, compiled Windows executables and a user guide are available in the electronic attachment and in the author's GitHub repository.

KEYWORDS

Font, JSON, PDF, Python, Script, Unicode

Vysadené pomocou balíčku `thesis` verzie 4.09; `https://latex.fekt.vut.cz/`

GMITTER, Jakub. *Skripty pro hromadnou úpravu fontů v PDF dokumentech*. Diplomová práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Vedúci práce: Ing. Pavel Hanák, Ph.D.

Vyhlásenie autora o pôvodnosti diela

Meno a priezvisko autora:	Bc. Jakub Gmitter
VUT ID autora:	220814
Typ práce:	Diplomová práca
Akademický rok:	2023/24
Téma závěrečnej práce:	Skripty pro hromadnou úpravu fontů v PDF dokumentech

Vyhlasujem, že svoju záverečnú prácu som vypracoval samostatne pod vedením vedúcej/cého záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podpisuje iba v tlačenej verzii.

POĎAKOVANIE

Rád by som poďakoval vedúcemu diplomovej práce pánovi Ing. Pavlovi Hanákovi, Ph.D., za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov

Úvod

Diplomová práca sa zameriava na vnútornú štruktúru PDF dokumentov, pričom špeciálny dôraz kladie na typy fontov, ich kódovanie a ToUnicode CMap tabuľky. Hlavným cieľom práce je vytvorenie skriptov, ktoré budú schopné detegovať chýbajúce ToUnicode mapovania a u vybraných fontov tieto mapovania dopĺňať. Prvý zo skriptov poskytne nástroj na poloautomatické spracovanie väčšieho množstva PDF dokumentov a zároveň bude kontrolovať, či ide o dokumenty, ktoré boli použité ako vzorky pre diplomovú prácu. Druhý zo skriptov bude opravovať poškodené kódovanie fontov a bude schopný upozorniť používateľov na potenciálne chyby, ktoré môžu vzniknúť počas tohto procesu. Oba skripty budú mať uložené kľúčové informácie potrebné pre ich fungovanie v samostatne vytvorených a prispôsobených JSON štruktúrach.

V rámci prvej kapitoly je popísaná vnútorná štruktúra PDF dokumentu. Kapitola obsahuje popis základných častí dokumentu a ich význam pri čítaní a zobrazovaní PDF dokumentu pomocou nástrojov na to určených.

Druhá kapitola je venovaná vysvetleniu reprezentácií jednotlivých strán dokumentov. Popisuje ich základné objekty, vysvetľuje ich význam a popisuje ich ukladanie do stromovej štruktúry.

V tretej kapitole sú vysvetlené rozdiely v rámci rôznych typov fontov, ktoré môžu byť v dokumentoch použité. Kapitola opisuje spôsoby vykresľovania znakov rôznymi fontami, históriu ich vývoja a spôsob ich kódovania. Taktiež sa v nej nachádza podrobný popis a význam objektov, ktoré fonty musia pre ich správnu reprezentáciu obsahovať.

Štvrtá kapitola opisuje vytvorený skript opravAR na overovanie súborov a JSON štruktúru, z ktorej skript čerpá informácie potrebné k overeniu PDF dokumentov. Približuje čitateľovi funkcie použité v rámci skriptu a ich implementáciu vo výslednom skripte.

Práca následne analyzuje vzorky, ktoré boli pri jej vytváraní použité. Uvádza ukážky konkrétnych chýb vo vzorkách pre lepšie priblíženie problematiky čitateľovi a spôsob, ktorým je možné docieliť opravu chybného kódovania. Taktiež uvádza ako rôzne poškodené sú dokumenty, ktoré práca analyzuje.

Na základe informácií vyplývajúcich z práce je popísaná tvorba opravného skriptu pod názvom Type1toUnicode. Skript opravuje poškodené kódovanie u PostScriptových fontov typu 1. Práca popisuje jeho priebeh a ukazuje najdôležitejšie časti, ktoré zabezpečujú jeho funkčnosť. Skript je prispôsobený pre bežného užívateľa a pri jeho tvorbe bolo dbané na čo najlepšiu informovanosť používateľa o priebehu skriptu a jednoduché používanie.

Vytvorené spustiteľné súbory skriptov a ich zdrojové kódy boli uverejnené prostredníctvom verejného GitHub repozitára autora. Zdrojové kódy obsahujú detailné komentáre, ktoré oboznamujú čitateľov s funkcionalitou skriptov. Zároveň bol vytvorený jednoduchý užívateľský návod na správne používanie skriptov.

Cieľom diplomovej práce je poskytnúť ucelený pohľad na problematiku, popísať metodiku a postupy použité pri vývoji skriptov aponúknuť užitočný popis pre ich efektívne využitie.

1 Štruktúra súboru PDF

Formát súboru PDF bol vyvinutý spoločnosťou Adobe v roku 1992. Cieľom tohto formátu je umožniť používateľom prezerať elektronické dokumenty nezávisle od aplikačného softvéru, hardvéru a operačného systému. PDF sa spolieha na rovnaký zobrazovací model ako PostScript, teda každý súbor obsahuje úplný popis dokumentu s pevne daným rozložením vrátane textu, fontov, obrázkov a ďalších informácií potrebných pre jeho zobrazenie.

Špecifikácia PDF bola bezplatne sprístupnená v roku 1993. Do roku 2008 však išlo o propiertárny formát spoločnosti Adobe. 1. júla 2008 bol vydaný ako otvorený štandard publikovaný ako ISO 32000-1:2008. Druhé vydanie PDF 2.0 bolo zverejnené v decembri 2020 a sprístupnené na bezplatné stiahnutie bolo v apríli 2023.[1]

Pre zjednodušenie sa PDF súbor skladá zo štyroch častí. Týmito časťami sú: hlavička, telo, tabuľka referencií (xref table), trailer.

Základné pojmy pre popis štruktúry:

- **Objekt** - základný dátový typ štandardu PDF. Existuje 9 typov objektov (null, boolean, integer, real name, string, array, dictionary a stream).
- **Slovník (dictionary)** - objekt páru kľúč - hodnota. Označuje sa « na začiatku a » na konci.
- **Nepriame objekty** - objekty, na ktoré sa odkazuje.
- **Priame objekty** - objekty, ktoré sú získané priamo.[1]

Obr. 1.1: Štruktúra PDF dokumentu

1.1 Hlavička

Prvým riadkom PDF súboru je hlavička. Označuje verziu PDF súboru. Formát záznamu je nasledovný: %PDF-(číslo verzie). Od verzie 1.4 však môže byť verzia

špecifikovaná záznamom Version v slovníku dokumentu catalog. Tento záznam sa nachádza v koreňovom zázname (Root) v časti Trailer. Ak je tento záznam prítomný, používa sa namiesto hlavičky. Toto umožňuje využívať funkcie novších verzií bez obavy z nemožnosti otvorenia dokumentu staršími PDF prehliadačmi.[2]

1.2 Telo

Telo súboru PDF pozostáva z nepriamych objektov, ktoré reprezentujú obsah dokumentu. Nepriame objekty na objekty iba odkazujú. Začínajú jedinečným identifikátorom objektu, ktoré umožňujú odkazovanie. Identifikátor sa skladá z čísla objektu a čísla generácie. Z iného miesta sa dá odkazovať pomocou nepriamej referencie, ktorá je tvorená číslom objektu, číslom generácie a kľúčovým slovom R (napríklad 10 2 R).

Číslo objektu je kladné celé číslo, ktoré identifikuje daný objekt. Číslo generácie umožňuje dokumenty revidovať a obnovovať ich. História revízií je uložená priamo v dokumente. Každý objekt preto obsahuje číslo generácie, ktoré začína nulou pri vytvorení dokumentu. Číslo sa vždy zvyšuje o 1 pri každej revízii objektu. Maximálna hodnota čísla generácie je 65536.[3]

1.3 Tabuľka referencií

Tabuľka referencií alebo tabuľka krížových odkazov obsahuje odkazy na všetky objekty v dokumente. Vďaka tejto tabuľke je možné náhodne pristupovať k objektom v súbore bez nutnosti prechádzania celého dokumentu. Každý objekt je reprezentovaný jednou položkou v tabuľke referencií.[4]

```
xref
0 49
0000000000 65535 f
0000212803 00000 n
0000212746 00000 n
0000212641 00000 n
0000206455 00000 n
0000003321 00000 n
0000000015 00000 n
```

Obr. 1.2: Ukážka tabuľky referencií

Riadky v tabuľke s dvomi číselnými hodnotami reprezentujú podsekciu v tabuľke. Prvé číslo zodpovedá číslu objektu a druhé číslo uvádza počet objektov v aktuálnej podsekcii. Objekt je vždy reprezentovaný záznamom o dĺžke 20 bajtov.

Prvých 10 bajtov (10 dekadických číslic) označuje bitový posun (offset) objektu. Posun objektu vyjadruje posun o určitý počet bajtov od začiatku dokumentu. Za posunom nasleduje číslo generácie objektu a písmeno f alebo n. Tieto písmená označujú, či je objekt voľný (f) alebo sa používa (n). Pokiaľ je objekt voľný, znamená to, že objekt môže byť stále prítomný v súbore, ale je označený ako voľný, takže by sa nemal používať. Objekty s príznakom f obsahujú odkaz na nasledujúci voľný objekt a číslo generácie, ktoré sa má použiť, ak sa objekt stane opäť platným. Príznak n je používaný pre označenie použitých a platných objektov.

Prvý objekt má vždy identifikačné číslo 0 a číslo generácie 65535. V PDF dokumentoch, ktoré boli postupne aktualizované, sa väčšinou nachádza viacero podsekcií. [4]

1.4 Trailer

Obsahuje umiestnenie tabuľky referencií a ďalšie špeciálne záznamy. Správny prehliadač dokumentov vždy číta PDF súbor od jeho konca a vďaka tomu dokáže rýchlo pristupovať k tabuľke referencií i k nepriamym objektom bez nutnosti analýzy celého súboru.[5]

```
trailer
<<  /Size 49
    /Root 1 0 R
    /Info 48 0 R
    /ID [<C8E8DC7ED4633085F4D1C4D680C27301>
         <82BBA9A63F8B076AF98E3A4B222E0132>]   >>
```

Obr. 1.3: Ukážka objektu Trailer

Trailer môže obsahovať tieto záznamy:

- **/Size** - počet záznamov v tabuľke referencií.
- **/Prev** - používa sa, iba pokiaľ dokument obsahuje viac ako jednu tabuľku referencií. Ide o bajtový posun od začiatku súboru po začiatok predchádzajúcej tabuľky referencií.
- **/Root** - určuje referenčný objekt pre katalóg dokumentu, ktorý obsahuje rôzne ukazovatele na iné objekty.
- **/Encrypt** - existuje, ak je PDF súbor zašifrovaný, ide o objekt s názvom Encryption.
- **/Info** - objekt obsahujúci všeobecné informácie o súbore (napríklad dátum poslednej modifikácie dokumentu).

- **/ID** - pole dvoch reťazcov, ktoré tvoria identifikátor súboru. Toto pole je povinné, ak existuje záznam Encrypt. [5]

2 Reprezentácia a obsah objektov strán

Jednotlivé strany sú v PDF súbore reprezentované ako súčasť stromovej štruktúry s názvom Pages. Každá strana je jedným uzlom stromovej štruktúry. Objekt Pages je koreňom tejto štruktúry. Každý PDF dokument môže obsahovať viacero takýchto štruktúr s rôznymi koreňmi.[6] Každý objekt Pages musí obsahovať nasledujúce položky:

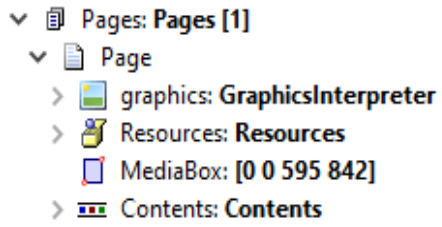
- **/Type** - typ objektu (v prípade koreňa musí byť typ Pages).
- **/Count** - vyjadruje počet objektov, ktoré sú potomkami objektu.
- **/Kids** - pomocou nepriameho objektu odkazuje na objekty, ktoré sú potomkami objektu.

Strany v PDF dokumente sú teda spojené pomocou stromovej štruktúry namiesto poľa. Vďaka tomuto usporiadaniu je možné strany rýchlejšie vyhľadávať. Dobré PDF aplikácie generujú vyvážený strom s minimálnou výškou uzlov.[6]

2.1 Objekt strany

Objekty typu Page, ktoré nemajú žiadneho potomka, predstavujú objekty jednotlivých strán v dokumente. Objekty typu Page môžu obsahovať nasledujúce položky:

- **/Type** - typ objektu (v tomto prípade Page).
- **/Parent** - obsahuje odkaz na objekt, ktorý je rodičom tohto uzla stromu.
- **/Contents** - ide o najdôležitejšiu položku objektu strany. Obsahuje inštrukcie o tom, čo sa má na stránku vykresliť.
- **/Rotate** - vyjadruje otočenie stránky v stupňoch. Pokiaľ je položka použitá, musí byť jej hodnota v násobkoch 90.
- **/Trans** - jedná sa o nepovinnú položku, ktorá sa využíva pri zobrazení strany v prezentácii. Jej hodnotou je definovaný štýl prechodu pri prechode z predchádzajúcej strany.
- **/Annots** - tento typ objektu umožňuje používateľom kliknúť na akčné tlačidlá a zároveň reprezentuje obsah vrátane videa, zvuku a 3D animácií .
- **/MediaBox** - rozmery obdĺžnika vyjadrené v predvolených jednotkách dĺžky, ktoré vymedzujú hranice fyzického média, na ktorom sa má stránka zobraziť alebo vytlačiť.
- **/CropBox** - rozmery obdĺžnika vyjadrené v predvolených jednotkách dĺžky, ktoré vymedzujú oblasť viditeľnú užívateľom. Keď je strana zobrazená alebo vytlačená, jej obsah musí byť orezaný do týchto rozmerov. Pokiaľ hodnota nie je zadefinovaná, jej hodnotou sú údaje v poli MediaBox.
- **/Resources** - používa sa na definovanie grafických objektov, ako sú fonty alebo obrázky. [7]



Obr. 2.1: Obsah objektu strany

2.1.1 Položka Contents

Tento objekt popisuje obsah strany. Pokiaľ by tento objekt na strane chýbal, strana sa zobrazí ako prázdna. Viditeľné prvky každej strany sú popísané v poli Contents. Jednotlivé položky poľa sú zreťazené a v tomto poradí sú interpretované pri zobrazení strany.

Obsah stránky popisuje pomocou špeciálnej syntaxe ktorý je odvodený z jazyka PostScript od spoločnosti Adobe.[8]

2.1.2 Položka Resources

Slovník Resources je základným komponentom PDF súboru. Pomáha udržiavať integritu a konzistenciu vizuálnych prvkov v dokumente. Zároveň zabezpečuje, že všetky potrebné zdroje sú dostupné. Pokiaľ by zdroj nebol zahrnutý v Resources, nie je možné ho použiť pri generovaní strany. Táto nedostupnosť môže viesť k problémom s vykresľovaním alebo k chýbaniu prvkov na výstupe. Môže obsahovať fonty, obrázky, farebné priestory a ďalšie externé údaje potrebné pre textový a grafický obsah strany.[9]

Typickým obsahom slovníka Resources sú objekty:

- **/Rozšírený stav grafiky (ExtGState)** - obsahuje informácie o stave grafiky, ako je šírka čiary, tvar čiary a informácie o farbe. Používa sa na zachovanie konzistencie vzhľadu grafických prvkov.
- **/Obrázky (XObjects)** - obsahuje zoznam obrázkov, ktoré sa na danej strane nachádzajú. Obsahuje napríklad názov obrázku, šírku, výšku, systém kódovania farieb a počet bitov reprezentujúcich jednotlivé farby pixelu.
- **/Farebný model (ColorSpace)** - určuje farebný model použitý pre obsah strany. Najčastejšie sa používajú modely RGB a CMYK. Je dôležitý pre správne podanie farieb najmä pri tlačení dokumentu.
- **/Fonty** - zoznam fontov použitých na danej strane dokumentu. Každý font použitý na strane tu musí byť uvedený s jeho jedinečným názvom a ďalšími parametrami, ktoré budú vysvetlené v ďalšej časti práce.[9]

3 Fonty

Počítačový font je digitálny dátový súbor tlačiteľných alebo zobrazovateľných typografických alebo textových znakov. Obsahuje sadu graficky súvisiacich glyfov. Každý glyf je špecifický tvar, dizajn a znázornenie znaku. Ide o konkrétne znázornenie prvku písaného jazyka v určitom fonte. Glyfom môže byť jedno písmeno, časť písmena (diakritika), špeciálny znak (interpunkcia), nepísaný znak (medzera) alebo kombinácia viacerých písmen. Všeobecne sa glyf vo výpočtovej technike môže označovať ako grafická jednotka.[10]

3.1 Bitmapové fonty

Bitmapový font využíva na reprezentáciu jednotlivých glyfov takzvané bitové mapy. Každá bitová mapa reprezentuje vo fonte iný glyf. Každý znak je popísaný v mriežke pixelov. Každý pixel teda presne určuje farbu na výstupnom zariadení.

Medzi najbežnejšie používané druhy bitmapových fontov patria napríklad SNF, PCF alebo BDF. Použitie bitmapového fontu prináša výhody najmä pri rýchlom načítaní fontu. Ich hlavnou nevýhodou je zlá škálovateľnosť. Vytvárajú sa podľa špecifickej veľkosti písma alebo rozlíšenia. Pri zmene veľkosti teda dochádza k strate kvality a čitateľnosti. Pre každú veľkosť je preto potrebné vytvoriť osobitnú bitovú mapu glyfov. Tieto boli populárne najmä v začiatkoch vývoja počítačov, kedy mali displeje nižšie rozlíšenia a nemohli zobrazovať zložitejšie obrazce.[11]

Obr. 3.1: Písmeno v bitmapovom fonte v mierke 8:1

3.2 Vektorové fonty

Často sa okrem označenia vektorové označujú aj ako obrysové fonty. Vektorové fonty používajú matematické výpočty na vytvorenie jednotlivých glyfov fontu. Na rozdiel od pixelov bitmapových fontov používajú množinu čiar a kriviek, ktoré môžu byť škálované bez toho, aby spôsobili stratu kvality.

Pre vektorové fonty je každý tvar reprezentovaný matematickým výrazom, ktorý v kombinácii s metrikou písma určujte štýl, tvar, výšku, šírku a priestor glyfu. Veľkosť glyfu a rotácia sa určuje pomocou informácií o zvislej mierke, ktoré sú uložené spolu s fontom.

Pri tlačení si tlačiareň vytvára bitovú mapu pre každý glyf v jej rozlíšení a otočí glyfy podľa pokynov.[12]

3.2.1 Bézierove krivky

Bézierove krivky sú matematický nástroj používaný pre tvorbu hladkých a presne definovaných kriviek. Krivky sú pomenované po francúzskom inžinierovi Pierrovi Bézierovi, ktorý ich používal v 60. rokoch 20. storočia pre dizajnovanie karosérií automobilov.

Krivka je definovaná pomocou kontrolných bodov. Množina kontrolných bodov P_0 až P_n, kde n sa nazýva rád kriky. Krivka vždy začína a končí v prvom a poslednom kontrolnom bode. Písmenom t sa označuje pokrok, ktorý urobil bod B.[13]

Bézierova krivka n-tého stupňa je definovaná pomocou vzťahu:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i, 0 \leq t \leq 1 \tag{3.1}$$

Lineárna Bézierova krivka

Lineárna krivka má dva kontrolné body, z čoho vyplýva, že n=1. Vzhľadom na to, že krivku tvoria iba dva body, sa prakticky jedná o priamku medzi bodmi P_0 a P_n.[13]

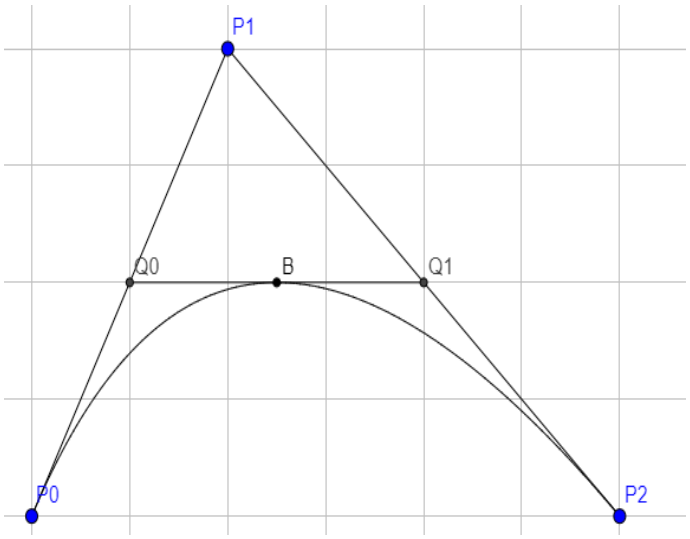
Kvadratická Bézierova krivka

Pri kvadratických Bézierových krivkách sa n=2. Je možné zostrojiť medziľahlé body Q_0 a Q_1 tak, že t sa mení od 0 do 1.

- Bod $Q_0(t)$ sa mení od P_0 do P_1 a opisuje lineárnu Bézierovu krivku.
- Bod $Q_1(t)$ sa mení od P_1 do P_2 a taktiež opisuje lineárnu Bézierovu krivku.
- Bod B(t) je interpolovaný lineárne od $Q_0(t)$ do $Q_1(t)$ a opisuje výslednú krivku.[13]

Kubická Bézierova krivka

Pre kubickú Bézierovu krivku sa n=3. Krivka začína v bode P_0, pokračuje smerom k P_1 a prichádza do P_3 v smere P_2. Krivka zvyčajne neprechádza cez body P_1 alebo P_2. Body slúžia pre smerové informácie o krivke.[13]



Obr. 3.2: Kvadratická Bézierova krivka

3.2.2 PostScriptové fonty

PostScriptové fonty typ 1 a typ 3 boli vytvorené spoločnosťou Adobe v roku 1984. Jednalo sa o jedny z prvých vektorových fontov a umožňovali vysokú kvalitu pri tlači na laserových tlačiarňach. Špecifikácia fontov nebola pôvodne zverejnená a jednalo sa o licencovaný produkt. Špecifikácia bola zverejnená v roku 1991 po predstavení vlastného typu fontov TrueType od spoločnosti Apple. Adobe ohlásilo v januári 2021 koniec podpory pre typ 1 od 1. 1. 2023.

Glyfy v PostScriptových fontoch sú popísané pomocou kubických Bézierových kriviek. Hlavným nedostatkom týchto fontov bola podpora iba 256 znakov v jednom súbore fontu. Toto obmedzenie bolo nevyhovujúce najmä pre jazyky s väčším počtom znakov alebo pre fonty, ktoré mali byť využité pre viacero jazykov. Štandardné kódovanie jednotlivých znakov je uvedené nižšie v tabuľkách 3.1 a 3.2.[14] [15]

Typ 3 oproti typu 1 dokáže napríklad určiť tieňovanie, farbu alebo vzory výplne. Pozostáva z glyfov definovaných pomocou úplného jazyka PostScript. Fonty typu 3 nie sú natívne podporované v žiadnej verzii Windows alebo macOS.[14] [15]

Tab. 3.1: Štandardné kódovanie pre PostScriptové fonty - 1. časť[16]

	0	1	2	3	4	5	6	7	8	9
0x	*NUL*	*SOH*	*STX*	*ETX*	*EOT*	*ENQ*	*ACK*	*BEL*	*BS*	*HT*
1x	*DLE*	*DC1*	*DC2*	*DC3*	*DC4*	*NAK*	*SYN*	*ETB*	*CAN*	*EM*
2x	*SP*	!	"	#	$	%	&	’	(	)
3x	0	1	2	3	4	5	6	7	8	9
4x	@	A	B	C	D	E	F	G	H	I
5x	P	Q	R	S	T	U	V	W	X	Y
6x	‘	a	b	c	d	e	f	g	h	i
7x	p	q	r	s	t	u	v	w	x	y
8x										
9x										
Ax		¡	¢	£	/	¥	f	§	¤	’
Bx		–	†	‡	·		¶	•	,	„
Cx			´	ˆ	˜		˘	˙	¨	
Dx	—									
Ex		Æ		ª					Ł	Ø
Fx		æ				ı			ł	ø

Tab. 3.2: Štandardné kódovanie pre PostScriptové fonty - 2. časť[16]

	A	B	C	D	E	F
0x	*LF*	*VT*	*FF*	*CR*	*SO*	*SI*
1x	*SUB*	*ESC*	*FS*	*GS*	*RS*	*US*
2x	*	+	,	-	.	/
3x	:	;	<	=	>	?
4x	J	K	L	M	N	O
5x	Z	[	\	]	ˆ	_
6x	j	k	l	m	n	o
7x	z	{	|	}	~	*DL*
8x						
9x						
Ax	“	«	‹	›	fi	fl
Bx	”	»	...	‰		¿
Cx		¸		˝	˛	ˇ
Dx						
Ex	Œ	º				
Fx	œ	ß				

3.2.3 TrueType fonty

Vznikli ako konkurencia pre PostScriptové fonty typu 1 od Adobe. Súbory obsahujúce fonty majú pre Windows koncovky názvu súboru .ttf alebo .tte a pre macOS majú príponu .dfont.

Na rozdiel od typu 1 však využívajú na opísanie glyfov kvadratické Bézierove krivky. Matematicky sú tieto krivky jednoduchšie a rýchlejšie na spracovanie ako kubické. Väčšina tvarov však vyžaduje na popísanie viac kontrolných bodov. Kvôli tomuto rozdielu je konverzia z typu 1 do TrueType stratová, opačná konverzia je však možná bez strát.[17]

V PDF súboroch môžu používať kódovania WinAnsiEncoding, MacRomanEncoding alebo vlastné kódovanie. Najčastejšie v PDF súboroch používajú práve WinAnsiEncoding. Toto kódovanie používajú s rozšíreniami, pretože pôvodné ANSI kódovanie obsahuje len 256 znakov a nepokrýva kompletne väčšinu svetových jazykov.[18]

3.2.4 OpenType fonty

Sú registrované pod ochrannou známkou spoločnosti Microsoft a vychádza z TrueType. Bežne sa používa na väčšine počítačových platforiem. Dokáže zvládnuť akékoľvek svetové písmo. V rámci jedného fontu môže byť obsiahnutých niekoľko svetových jazykov.

Okrem rôznych písmen obsahujú fonty aj špeciálne znaky alebo rôzne veľkosti číslic (indexy, exponenty). Každý font OpentType môže obsahovať maximálne 65536 glyfov. Pokiaľ do fontu nie sú pridané žiadne iné ako základné glyfy, môžu byť OpenType fonty podstatne menšie ako fonty typu 1.[19]

3.3 Fonty založené na ťahoch

V angličtine sa fonty označujú názvom stroke-based fonts. Od bitmapových a vektorových fontov sa líšia pomocou definície jednotlivých glyfov fontu. Vektorové fonty popisujú glyfy pomocou Bézierovych kriviek, zatiaľ čo fonty založené na ťahoch popisujú glyfy pomocou série ťahov, ktoré vedú k zobrazeniu glyfu.

Hlavnou výhodou týchto fontov je zníženie počtu kontrolných bodov oproti vektorovým fontom. Zároveň umožňuje napríklad použitie rovnakých bodov na generovanie glyfov s rôznou šírkou. Pre rozdielnu šírku glyfu sa využijú rovnaké kontrolné body, dôjde len k pozmeneniu hrúbky čiary pre vykreslenie. Vďaka tomuto kroku je možné ušetriť na celkovej veľkosti súborov písiem. Pre zariadenia s obmedzeným úložným priestorom, ako napríklad IoT zariadenia, sú tieto fonty vďaka menšej veľkosti ideálnou voľbou.

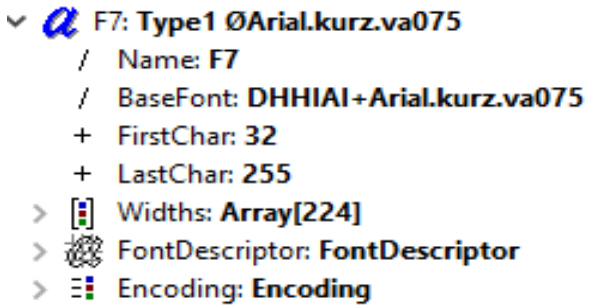
U fontov založených na ťahu je možné vyplniť rovnaké dráhy ťahu rôznymi profilmi, čo vedie k rôznym vizuálnym tvarom bez toho, aby bolo potrebné špecifikovať polohy kontrolných bodov každého obrysu.[20]

3.4 Reprezentácia fontu v PDF súbore

Objekty fontov použitých na jednotlivých stranách v PDF súbore sa nachádzajú v slovníku Font, ktorý je súčasťou Resources. Každá stránka má vlastný slovník fontov. Základnými prvkami sú identifikátor znaku (CID) a identifikátor glyfu (GID). Spolu tvoria dvojice, pričom každému CID zodpovedá práve jedno GID.[1]

Základný obsah objektu fontu je:

- **/Type** - označuje typ objektu. Táto položka je povinná a vždy musí obsahovať hodnotu Font.
- **/Subtype** - určuje, o aký druh fontu ide. Najčastejšie druhy fontu sú TrueType alebo Type1.
- **/BaseFont** - názov fontu jedinečný pre celý dokument. Pod týmto názvom sa fonty zobrazujú v prehliadačoch dokumentov. Pre predchádzanie rovnakých názvov fontov sa hlavne pri type 1 pridáva pred názov fontu 6 náhodných písmen a od názvu sa oddeľujú znakom plus. Názov fontu môže byť napríklad DHHIAI+Arial.kurz.va075, ako je uvedené na obrázku 3.3.
- **/Name** - táto položka nie je povinná. Určuje jedinečný názov fontu v rámci jednej strany dokumentu.
- **/FirstChar** - dekadická hodnota CID prvého znaku uvedeného v poli šírky znakov.
- **/LastChar** - dekadická hodnota CID posledného znaku uvedeného v poli šírky znakov.[1]



Obr. 3.3: Štruktúra objektu fontu

Pole šírky znakov

V objekte fontu sa nachádza pole definujúce šírku jednotlivých glyfov fontu. Pole sa nazýva Widths. Každý prvok poľa predstavuje šírku glyfu pre CID, ktoré sa rovná hodnote FirstChar plus index poľa. Pre CID mimo rozsahu FirstChar až LastChar sa pre daný font použije hodnota MissingWidth z položky FontDescriptor. Šírka glyfu sa meria v jednotkách, pričom 1000 jednotiek zodpovedá jednej jednotke textového priestoru. Tieto šírky musia byť v súlade so skutočnými šírkami uvedenými v špecifikácii fontu.[1]

Popisovač písma

Objekt popisu písma (FontDescriptor) uchováva ďalšie informácie o tvare glyfov, iné ako ich šírku. Základnou položkou je názov fontu (FontName), ktorý musí zodpovedať položke BaseFont, aby bol správne priradený.[1]

Položka Flags je 32-bitová položka vyjadrujúca celé kladné číslo. Najmenej významný bit čísla je označovaný ako prvý bit. Bity s hodnotou 1 vyjadrujú podľa pozície nasledujúce informácie:

- **1. bit** - všetky glyfy fontu majú rovnakú šírku.
- **2. bit** - glyfy majú pätky, čo sú krátke ťahy nakreslené pod uhlom na vrchnej a spodnej časti stonky glyfu.
- **3. bit** - font obsahuje glyfy mimo štandardnej znakovej sady latinky.
- **4. bit** - glyfy sú písané v kurzíve.
- **6. bit** - font používa štandardnú znakovú sadu latinky Adobe alebo jej podmnožinu.
- **7. bit** - glyfy majú naklonené dominantné vertikálne ťahy.

Všetky nedefinované bity musia byť nastavené na hodnotu 0.[21]

Ďalšou dôležitou položkou popisovača je FontBBox. Táto položka je povinná pre všetky typy písma okrem PostScriptového Type 3. Pomocou štyroch bodov vyjadruje obdĺžnik v súradnicovom systéme glyfov, ktorý určuje ohraničenie písma. Ide o najmenší obdĺžnik obklopujúci tvar, ktorý by vznikol, pokiaľ sa všetky glyfy umiestnia tak, aby začínali v rovnakom bode.[1]

Nasledujúce položky popisujú ďalšie špecifikácie glyfov vo fonte:

- **ItalicAngle** - uhol dominantných vertikálnych ťahov písma vyjadrený v stupňoch proti smeru hodinových ručičiek. Hodnota je teda záporná pre všetky fonty so sklonom glyfov doprava, takže takmer pre všetky písma s kurzívou.
- **Ascent** - maximálna hodnota výšky glyfu nad základnou čiarou vo fonte. Výška glyfov s diakritikou sa neberie do úvahy.
- **Descent** - maximálna hodnota výšky glyfu pod základnou čiarou vo fonte. Môže dosahovať zápornú hodnotu alebo nulu.

- **CapHeight** - vertikálna súradnica vrchnej časti veľkých písmen meraná od základnej čiary.
- **SteamV** - hrúbka hlavných vertikálnych čiar glyfov vo fonte meraná horizontálne.

Položky pre popis kódovania

Kódovanie fontu je definované v položke Encoding. Špecifikuje kódovanie fontu, pokiaľ sa líši od zabudovaného kódovania. Hodnota položky je buď názov preddefinovaného kódovania (napríklad WinAnsiEncoding), alebo slovník kódovania, ktorý špecifikuje rozdiely od zabudovaného kódovania. Tieto rozdiely môžu byť popísané v poli rozdielov (Differences). V tomto poli je možné nájsť všetky takto rozdielne glyfy podľa ich hodnoty GID.

```
'
/CIDInit /ProcSet findresource begin 12 dict begin begincmap /CIDSystemInfo <<
/Registry (PAGE1+F6+0) /Ordering (T1UV) /Supplement 0 >> def
/CMapName /PAGE1+F6+0 def
/CMapType 2 def
1 begincodespacerange <01> <09> endcodespacerange
9 beginbfchar
<01> <004E>
<02> <00C1>
<03> <0160>
<04> <0020>
<05> <0052>
<06> <004F>
<07> <005A>
<08> <0048>
<09> <0056>
endbfchar
endcmap CMapName currentdict /CMap defineresource pop end end
'
```

Obr. 3.4: Ukážka objektu ToUnicode

Pokiaľ sa vyžaduje zmena kódovania konkrétnych znakov do Unicode, sú tieto údaje uvedené v objekte názvu ToUnicode. Objekt mapuje pomocou mapy charakterov jednotlivé CID na hodnoty Unicode. Ukážkovú mapu je možné vidieť na obrázku 3.4.[1]

3.5 Štandard Unicode

Unicode je globálny štandard, ktorý umožňuje počítačom reprezentovať a manipulovať textom zo všetkých písomných systémov sveta. Poskytuje jednotný súbor kódov pre všetky znaky, nezávisle od platformy, programu alebo jazyka. To znamená, že

text kódovaný v Unicode je čitateľný a použiteľný na rôznych zariadeniach a v rôznych aplikáciách. Podporuje prakticky všetky jazyky a písma, vrátane diakritiky, hieroglyfov, čínskych, japonských alebo kórejských znakov.

Unicode je navrhnutý tak, aby bol rozšíriteľný, čo znamená, že nové znaky a symboly (napríklad nové emoji alebo znaky pre nové písma) môžu byť pridané bez narušenia existujúcej štruktúry. Vďaka univerzálnej prijateľnosti a kompatibilite, môžu byť dáta ľahko zdieľané a spracovávané v rôznych systémoch a aplikáciách bez rizika stratenia alebo skreslenia informácií.[22]

4 Oprava kódovania

4.1 Skript pre overenie súboru

Prvý z vytvorených skriptov v rámci práce zabezpečuje overenie PDF súboru. Overuje, či je analyzovaný súbor naozaj nepozmeneným originálnym vydaním jedného z časopisov Amatérskeho rádia a jeho ďalších titulov: Praktická elektronika, Konstrukční elektronika a Stavebnice a konstrukce. Skript najskôr vypočíta heš SHA-256 z PDF súboru a následne hľadá zhodu hešu s hešmi v JSON štruktúre, ktorá bola vytvorená na tieto účely a manuálne naplnená hodnotami hešov pre všetky analyzované časopisy v rámci diplomovej práce. Vďaka tomuto porovnaniu skript preskočí všetky neoriginálne verzie časopisov a prípadné ďalšie PDF súbory nachádzajúce sa v rovnakom adresári.

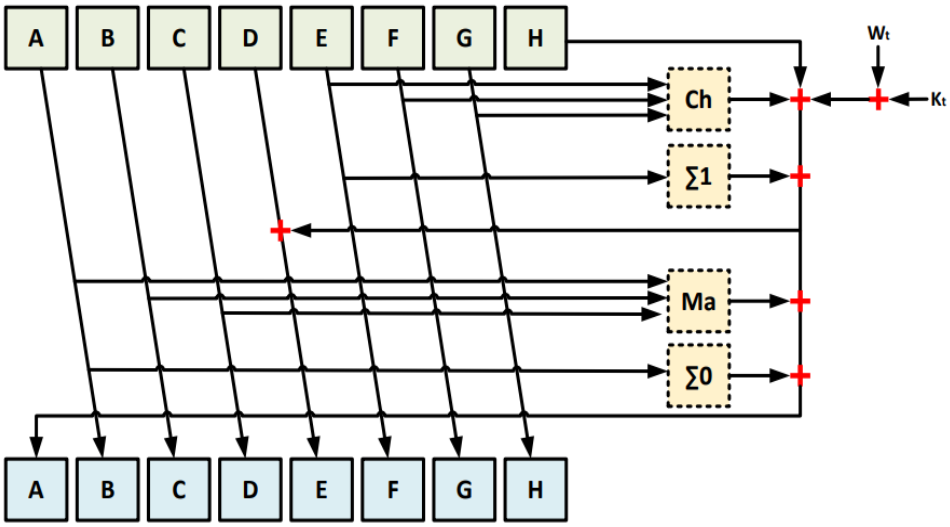
4.1.1 SHA-256

Hešovacia funkcia je algoritmus, ktorý preberá vstupné dáta ľubovoľnej dĺžky a generuje z nich heš, teda výstupnú hodnotu s pevnou dĺžkou. Tento heš slúži ako digitálny odtlačok alebo podpis vstupných dát. Hešovacie funkcie sú základnou súčasťou počítačovej bezpečnosti, databáz, overovania pravosti dát a ďalších aplikácií, kde je potrebná rýchla a efektívna identifikácia a verifikácia informácií. Ide teda o kompresnú funkciu. Hešovacia funkcia vždy generuje rovnaký heš pre rovnaké vstupné dáta bez ohľadu na to, koľkokrát je funkcia aplikovaná. Malá zmena vstupu (napríklad zmena jedného bitu) by mala viesť k úplne odlišnému hešu. Tento efekt zabezpečuje, že hešovacie funkcie sú odolné voči pokusom o predvídanie hešovaných hodnôt alebo útokom, ktoré sa snažia nájsť dva podobné vstupy generujúce rovnaký heš. Od hešovacích funkcií sú vyžadované tieto vlastnosti:

- **jednosmernosť** - pre daný heš h je prakticky nemožné nájsť nejaký iný vstup V, ktorého heš by odpovedal hodnote hešu h.
- **odolnosť voči modifikácii vzoru** - pre daný heš h = H(V1) je nemožné nájsť V2 také, aby H(V2) = h.
- **odolnosť voči kolízii** - je nemožné nájsť dva rôzne vstupy V1 a V2, pre ktoré by platilo H(V1) = H(V2).[23]

Hešovacia funkcia SHA-256 spadá do rodiny SHA-2, ktorá bola prvýkrát publikovaná v roku 2001. SHA-2 pozostáva zo šiestich hešovacích funkcií: SHA-224, SHA-256, SHA-384, SHA-512 SHA-512/224 a SHA-512/256. Číselné hodnoty v názvoch udávajú veľkosti výstupu funkcie v bitoch. Výstup funkcie je interpretovaný v hexadecimálnom formáte.

Hešovacie funkcie ako SHA-256 sú základným stavebným kameňom v mnohých aspektoch informačnej bezpečnosti vrátane digitálnych podpisov, overenia integrity dát a zabezpečenia databáz. SHA-256 je zvlášť obľúbený kvôli svojej odolnosti voči útokom (napríklad odolnosť voči kolízii).[24]



Obr. 4.1: Iterácia v SHA-256

Jednotlivé operácie v jednej iterácii SHA-256:

- $Ch(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$
- $\sum_1(E) = (E >>> 6) \oplus (E >>> 11) \oplus (E >>> 25)$
- $Ma(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$
- $\sum_0(A) = (A >>> 2) \oplus (A >>> 13) \oplus (A >>> 22)$
- operácia + vyjadruje sčítanie modulo 2^{32} [24]

SHA-256 je preferovaný pred MD5 kvôli vyššej bezpečnosti, ktorú poskytuje. Heš SHA-256 má dlhší výstupný reťazec (256 bitov oproti 128 bitom u MD5), čo ho robí odolnejším voči kolíziám a útokom. Zatiaľ čo MD5 je zraniteľný voči kolíznym a pre-image útokom, ktoré sú už prakticky realizovateľné, SHA-256 zatiaľ odoláva známym útokom. Navyše, SHA-256 je súčasťou rodiny SHA-2, ktorá je považovaná za priemyselný štandard a je odporúčaná pre použitie v moderných bezpečnostných aplikáciách. Zvýšená odolnosť voči kolíziám je vyžadovaná pri analýze veľkého počtu PDF súborov. Pokiaľ by k takejto kolízii došlo, mohol by byť opravný skript spustený pre PDF súbor, ktorý neodpovedá vzorovým časopisom použitým v diplomovej práci.

4.1.2 JSON štruktúra pre overovanie súborov

Pre potreby uchovania hešov vytvorených z analyzovaných PDF súborov bola vytvorená JSON štruktúra, ktorá ich prehľadne ukladá a rozdeľuje podľa názvu časopisu, rokov vydania časopisu a čísla vydania v rámci daného kalendárneho roku. Ukážka štruktúry so vzorovými údajmi je uvedená vo výpise 4.1.

Štruktúra má na svojom začiatku definovaných niekoľko kľúčov, ktoré o nej udávajú potrebné informácie. Tieto kľúče sú:

- **product** - popis produktu, v tomto prípade ide o jeho pomenovanie.
- **version** - verzia súboru (verzia sa mení po doplnení hešov pre ďalšie vydania časopisov).
- **releaseDate** - dátum vydania JSON súboru v ISO formáte.

Výpis 4.1: Vzor dát v JSON štruktúre pre overovanie súborov

```
1  {
2  "product": "hashes␣of␣magazines",
3  "version": 1.0,
4  "releaseDate": "2024-04-22T00:00:00.000Z",
5  "magazines": {
6  "Konstrukcni␣elektronika": {
7      "name": "Konstrukcna␣elektronika",
8      "year": {
9          "2000": {
10         "1": "6753608273b52995a7d1660891d6144d18b4e63",
11         "2": "0e1336475cc77f536856de5941f7f14c9241fcf",
12         }
13     }
14 },
15 "Prakticka␣elektronika": {
16     "name": "Prakticka␣elektronika",
17     "year": {
18         "2000": {
19         "1": "424dff7f308d46a706ee7089a08f548d24b29c444",
20         "2": "5c4623c4e09abd1c6f4288f836d279ac2a66287f5",
21         "3": "0da64f2a167043637d5fe4171a639304b61601291",
22         }
23     }
24   }
25 }
26 }
```

Po týchto údajoch nasleduje objekt magazines, ktorý v sebe obsahuje informácie o rôznych časopisoch. Pre správnu funkciu overovacieho skriptu je potrebné, aby všetky informácie o prípadných ďalších pridaných časopisoch boli uložené práve do tohto objektu. Každý časopis by mal obsahovať kľúč s jeho názvom. Tento kľúč pre funkciu samotného skriptu nie je povinný a má význam iba pre lepšiu prehľadnosť JSON súboru. Povinným objektom v súbore je year. Tento objekt obsahuje ďalšie objekty, ktoré konkretizujú roky vydania pre daný časopis. V každom roku vydania sa potom nachádzajú kľúče, ktorých názov definuje číslo vydania časopisu v danom roku. K tomuto číslu je potom priradená vygenerovaná hodnota hešu SHA-256 pre daný PDF súbor. Heš bol vypočítaný z originálnych elektronických vydaní časopisov, ktoré slúžili ako vzorové dáta pre diplomovú prácu. Heš sa počíta iba z obsahu súboru, a teda rôzne pomenovanie PDF súboru nemá vplyv na výsledný heš. JSON je preto možné kedykoľvek rozšíriť za predpokladu dodržania štruktúry nielen o ďalšie ročníky vydania, ale aj iné časopisy, ktoré nie sú zhodné so vzorovými dátami. K vypočítaniu hešu SHA-256 PDF súboru je možné využiť aj nástroj certutil, ktorý je súčasťou inštalácie operačného systému Windows. Pre výpočet SHA-256 jedného PDF súboru je potrebné do príkazového riadku zadať príkaz:

certutil -hashfile {názov súboru}.pdf sha256

Na hromadný výpočet hešov SHA-256 vo Windows pomocou nástroja certutil je možné použiť nasledujúci príkaz v príkazovom riadku:

forfiles /m *.pdf /c "cmd /c certutil -hashfile @file sha256"

- **forfiles** - príkaz sa používa na vykonávanie operácií na súboroch v priečinku. Je užitočný na spracovanie veľkého množstva súborov naraz.
- **/m** - prepínač určuje masku súborov, na ktoré sa bude príkaz vzťahovať. V tomto prípade je maska *.pdf, čo znamená, že príkaz sa aplikuje na všetky súbory s príponou .pdf v aktuálnom priečinku.
- **/c** - prepínač špecifikuje príkaz, ktorý sa má vykonať pre každý nájdený súbor.

4.1.3 Popis skriptu pre overovanie súborov

Pre overenie súborov bol v rámci diplomovej práce vytvorený skript v jazyku Python, ktorého úlohou je porovnať heše vstupných súborov so záznamami v JSON súbore. Skript prechádza všetky PDF dokumenty, ktoré sú umiestnené v ceste zadanej užívateľom pomocou argumentu pri spúšťaní skriptu. Pokiaľ zhodu nájde, spustí ako ďalší proces .exe súbor, ktorý opravuje kódovania fontov v súbore. Popisu tohto spustiteľného súboru sa práca venuje v ďalších častiach.

Skript importuje pre svoju funkcionalitu niektoré funkcie z knižníc: os, subprocess, json, hashlib, argparse a colorama. Knižnica colorama je potrebná pre správnu interpretáciu farebného textu v príkazovom riadku. V rámci práce bola použitá kniž-

nica vo verzii 0.4.6. Ostatné knižnice sú súčasťou čistej inštalácie Pythonu, a teda ich nie je potrebné dodatočne inštalovať. Konkrétne funkcie sú uvedené vo výpise 4.2.

Výpis 4.2: Importovanie potrebných knižníc pre overovací skript

```
1 from os import listdir, path
2 from subprocess import run, CalledProcessError
3 from json import load, JSONDecodeError
4 from hashlib import sha256
5 from argparse import ArgumentParser
6 from colorama import init
```

Vyhľadávanie zabezpečuje funkcia s názvom find_hash_in_json. Funkcia je definovaná s tromi vstupnými parametrami:

- **json_file_path** - cesta, ktorá definuje umiestnenie JSON súboru s hešmi v rámci súborového systému.
- **dir_path** - cesta k adresáru, v ktorom skript vyhľadáva PDF súbory pre analýzu.
- **verbose** - parameter, ktorý určuje, či sa bude spúšťať opravný skript v móde s rozšírenými logovacími výpismi.

Funkcia najskôr načíta do premennej data obsah JSON súboru. Súbor otvára len v režime čítania. Nakoľko needituje dodanú JSON štruktúru, vyššie oprávnenia ako na čítanie nepotrebuje. V rámci súborového systému musí mať užívateľ definované minimálne právo čítania vo zvolenom JSON súbore.

Po načítaní dát sa inicializuje premenná json_hashes ako prázdna množina. Množina je v Pythone dátová štruktúra, ktorá obsahuje jedinečné prvky, a teda sa každý prvok môže v jednej množine vyskytovať iba raz. Množina zabraňuje duplicite hodnôt v nej obsiahnutých. Pokiaľ by bol pokus vložiť do množiny prvok, ktorý už obsahuje, Python ho znovu nevloží.

Naplnenie premennej json_hashes samotnými hešmi z JSON súboru prebehne vďaka definovanej funkcii collect_hashes. Jedná sa o rekurzívnu funkciu, ktorá prechádza slovníkom a zhromažďuje hodnoty do množiny json_hashes. Pomocou cyklu for prechádza iteratívne slovník z parametru hodnoty. V cykle predstavuje month kľúč a value hodnotu spojenú s týmto kľúčom. V cykle sa testuje, či je premenná value slovníkom. Pokiaľ áno, znamená to, že dáta obsahujú ďalšiu úroveň vnorenia a funkcia rekurzívne zavolá samu seba so vstupným parametrom value. Týmto spôsobom je zabezpečené, že funkcia môže prechádzať dáta do akejkoľvek hĺbky vnorenia. Pokiaľ už v niektorej z iterácii nie je value slovníkom, ale konkrétnou hodnotou, pridáva sa táto hodnota metódou add do množiny json_hashes. Následne sa funkcia volá v ďalšom for cykle, ktorý prechádza všetky objekty magazines v JSON súbore. Pre každý časopis v JSON súbore je teda volaná funkcia collect_hashes

zvlášť a všetky hodnoty kľúčov, ktoré objekty obsahujú, sa zhromaždia do jednej premennej. Týmto postupom sú heše efektívne pripravené na porovnávanie s hešmi PDF súborov, ktoré sa budú analyzovať a opravovať.

Výpis 4.3: Funkcia pre nájdenie hešu v JSON

```
1  @classmethod
2  def find_hash_in_json(cls, json_file_path, dir_path, verbose):
3      try:
4          with open(json_file_path, 'r') as file:
5              data = load(file)
6          json_hashes = set()
7          def collect_hashes(year_dict):
8              for month, value in year_dict.items():
9                  if isinstance(value, dict):
10                     collect_hashes(value)
11                 else:
12                     json_hashes.add(value)
13         for magazine in data["magazines"].values():
14             collect_hashes(magazine)
15         for filename in listdir(dir_path):
16             if filename.endswith(".pdf"):
17                 file_path = path.join(dir_path, filename)
18                 with open(file_path, 'rb') as file:
19                     file_content = file.read()
20                     file_hash=sha256(file_content).hexdigest()
21                     if file_hash in json_hashes and verbose:
22                         run(['Type1toUnicode.exe', '-p',
23                         file_path, '-f', 'to_unicode.json',
24                         '-v'], check=True)
25                     elif file_hash in json_hashes:
26                         run(['Type1toUnicode.exe', '-p',
27                         file_path, '-f', 'to_unicode.json'],
28                         check=True)
29                     else:
30                         print(f"\033[33mHash␣not␣found␣for
31 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣file:␣{filename}\033[0m")
32     except FileNotFoundError:
33         print("\033[31mFile␣does␣not␣exist.\033[0m")
34     except JSONDecodeError as e:
35         print("\033[31mError␣in␣parsing␣JSON␣file:
36 ␣␣␣␣␣␣␣␣{json_file_path}\n{e}\033[0m")
37     except CalledProcessError:
```

```
38          print("\033[31mError␣when␣running
39 ␣␣␣␣␣␣␣␣Type1toUnicode.exe\033[0m")
40      except Exception as e:
41          print(f"\033[31mAn␣exception␣occurred:␣{e}\033[0m")
```

V poslednej časti funkcia prechádza všetky súbory v adresári zadanom užívateľom pomocou argumentu pri volaní skriptu. Pracuje iba sú súbormi, ktoré majú koncovku .pdf, a všetky ostatné súbory v adresári preskakuje. Pokiaľ súbor s koncovkou .pdf nájde, uloží celú cestu k nemu vrátane názvu súboru do premennej file_path. Tento súbor otvára v binárnom režime, čo je nutné pre vypočítanie hešu SHA-256 z obsahu súboru. Celý obsah načíta do pamäti v premennej file_content a vypočíta z neho heš. Tento heš prevádza na hexadecimálny tvar, nakoľko všetky heše v JSON súbore sú uložené v hexadecimálnom tvare. Pokiaľ je hodnota hešu zhodná s niektorou z hodnôt v json_hashes, spustí opravný skript. Skript spustí buď v režime s klasickým, alebo s rozšíreným logovaním, výber záleží na argumentoch zadaných užívateľom v príkazovom riadku. Pokiaľ sa heš nezhoduje so žiadnym záznamom v JSON súbore, skript vypíše do konzoly upozornenie, v ktorom uvedie, pre ktorý PDF súbor nenašiel zhodu. Tento výpis je pre prehľadnejšie upozornenie užívateľa vypísaný žltou farbou.

Funkcia kontroluje aj existenciu všetkých používaných súborov, správnu syntax JSON súboru, možnosť spustenia skriptu pre opravu fontov a všeobecne ďalšie výnimky, ktoré môžu nastať. Pri chybe v syntaxi JSON upozorní užívateľa, kde sa chyba v súbore nachádza. Tieto chybové stavy sú do konzoly vpisované červenou farbou. Zdrojový kód funkcie je uvedený vo výpise 4.3.

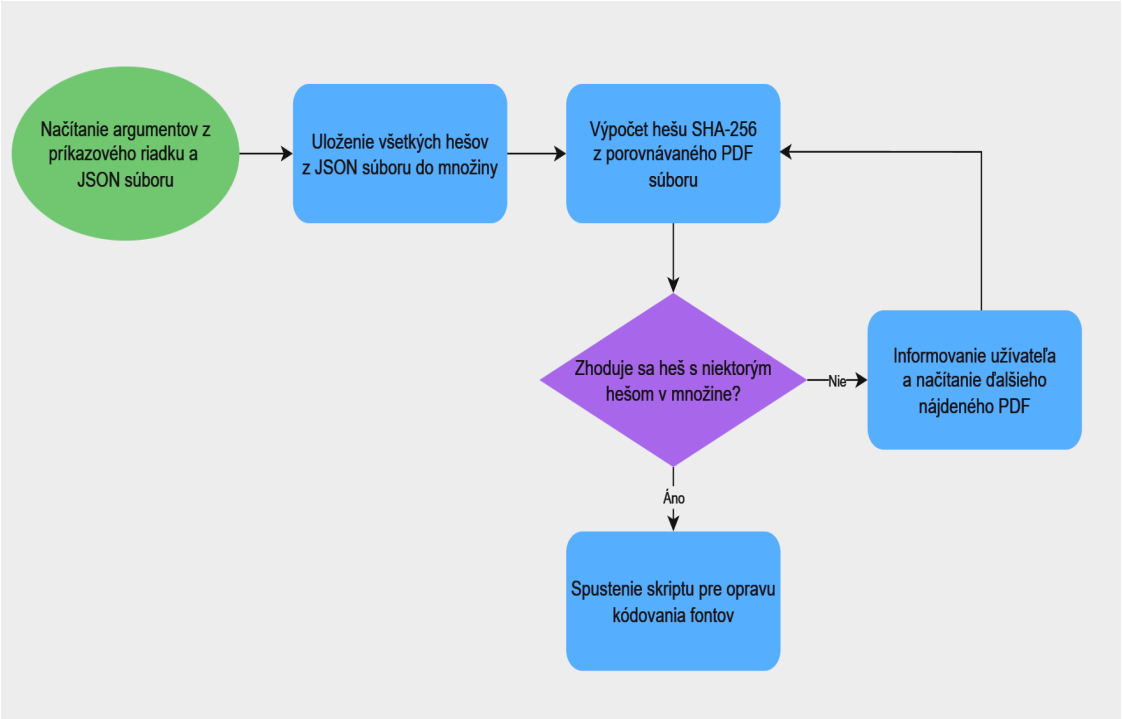
V hlavnej funkcii main sa spracovávajú argumenty zadané užívateľom do príkazového riadku. Na základe týchto argumentov sa nastavia parametre pre volanie funkcie find_hash_in_json. V skripte sú definované nasledujúce tri argumenty:

- **-d (–pdf_dir)** - umožňuje užívateľovi špecifikovať adresár, v ktorom bude skript vyhľadávať PDF súbory. Tento argument je povinný a očakáva reťazec reprezentujúci cestu k adresáru.
- **-j (–json_file)** - umožňuje užívateľovi špecifikovať cestu k JSON súboru, obsahujúcemu heše. Takisto ide o povinný argument.
- **-v (–verbose)** - argument typu boolean. Ak je prítomný, nastaví svoju hodnotu na True. Tento prepínač je použitý na ovládanie úrovne logovania. Pokiaľ je nastavený, nastavuje logovanie opravného skriptu na rozšírenú úroveň.

Výpis 4.4: Funkcia main opravného skriptu

```
def main():
    init()
    argparser = ArgumentParser()
    argparser.add_argument('-d', '--pdf_dir', type=str,
    required=True,
    help='Defines␣the␣path␣to␣directory␣with␣.pdf␣files')
    argparser.add_argument('-j', '--json_file', type=str,
    required=True, help='Defines␣the␣path␣to␣the␣.json␣file')
    argparser.add_argument('-v', '--verbose',
    action='store_true', help='Enable␣verbose␣output
␣␣␣␣(prints␣more␣information)')
    args = argparser.parse_args()
    Hash.find_hash_in_json(args.json_file, args.pdf_dir,
    args.verbose)

if __name__ == '__main__':
    main()
```



Obr. 4.2: Vývojový diagram skriptu pre overenie súboru

Vývojový diagram skriptu je uvedený na obrázku 4.2. Argumenty sa ukladajú do premennej args, kde budú mať atribútry pdf_dir, json_file a verbose. Po uložení

argumentov sa volá metóda find_hash_in_json za pomoci zadaných argumentov. Obsah funkcie main je vo výpise 4.4. Celý zdrojový kód skriptu vrátane komentárov priamo v kóde je ku práci priložený pod názvom opravAR.

4.2 Analýza vzoriek

V rámci práce bolo potrebné analyzovať vzorové súbory a následne opraviť kódovanie fontov. Vzorové súbory pochádzajú z elektronických verzií vydaní časopisu Amatérske rádio a jeho ďalších titulov: Praktická elektronika, Konštrukční elektronika a Stavebnice a konstrukce.

Pri vzorových súborov dochádza buď ku chybnému kódovaniu znakov s diakritikou, alebo k zlému kódovaniu všetkých znakov. Túto chybu nie je pri bežnom prehliadaní možné pozorovať. Chybu je možné pozorovať až pri kopírovaní textu z originálneho súboru, napríklad do poznámkového bloku. Chyba vzniká v dôsledku chýbajúcich objektov pre správne kódovanie do Unicode. V analyzovaných časopisoch sa najčastejšie jedná o chýbajúci objekt ToUnicode. Pokiaľ tento objekt chýba, prehliadač dokumentu pri kopírovaní textu používa hodnotu CID pre kódovanie do Unicode. Prehľad druhov chýb rozdelený podľa rokov vydania je uvedený v tabuľke 4.1.

Tab. 4.1: Druhy chýb vo vzorových súboroch

	Amatérske rádio	Praktická elektronika	Konstrukční elektronika	Stavebnice a konstrukce
Fonty bez chyby		1999, 2016	1999	
Chyby pri znakoch s diakritikou	1999 až 2004	1997, 1998	1997, 1998	1999, 2001, 2002
Chyby pri všetkých znakoch		2000 až 2007, 2018 až 2022	2000 až 2007	

Pre potreby diplomovej práce boli využité vydania časopisov Praktická elektronika a Konstrukční elektronika z rokov 2000 až 2007 a 2018 až 2021. Celkovo ide o 192 vzoriek.

Príklad chybného kódovania je možné ukázať na slove „výrobky“ z tretieho čísla časopisu Praktická elektronika z roku 2001:

- Slovo v pôvodnom súbore: výrobky
- Slovo po kopírovaní do poznámkového bloku: -&——

Chybný preklad je zapríčinený nesprávnym kódovaním znakov do štandardu Unicode. Unicode je univerzálne kódovanie akýchkoľvek elektronických znakov. Tento štandard využívajú všetky bežne používané operačné systémy. Unicode používa pre reprezentáciu znaku 8 alebo 16 bitové číslo (v závislosti od verzie). V rámci práce je číslo vždy uvádzané v 16 bitovom hexadecimálnom tvare s reťazcom U+ pred hexadecimálnym číslom.

Tab. 4.2: Kódovanie ukážkového slova

Znak	Reprezentácia znaku pri kopírovaní	Správne kódovanie podľa Unicode	CID znaku vo vzorke	GID znaku vo vzorke	GID hexadecimálne
v	U+0015	U+0076	21	118	76
ý	U+0026	U+00FD	38	253	FD
r	U+0003	U+0072	3	114	72
o	U+0008	U+006F	8	111	6F
b	U+0010	U+0062	16	98	62
k	U+001A	U+006B	26	107	6B
y	U+0011	U+0079	17	121	79

Hodnota CID sa môže v rámci rôznych fontov meniť. Je generovaná podľa prvého výskytu daného znaku v súbore. Pokiaľ je teda prvé písmeno fontu použité v súbore „A“, bude mať hodnotu CID 1. GID však zostáva vo vzorových súboroch nezmenené pri všetkých fontoch. V tabuľke si možno všimnúť vzťah medzi GID a správnym Unicode kódovaním, kde ide o prevod decimálnej hodnoty GID na hexadecimálnu a výplň hodnoty nulami. Niektoré prehliadače dokumentov ako napríklad Infix PDF editor alebo Evince dokážu vďaka tomuto vzťahu čiastočne opraviť chybné kódovanie. Pri kopírovaní z týchto prehliadačov sa teda kódujú základné písmená a číslice správne. Prehliadače však nevedia vygenerovať a uložiť do súboru objekt ToUnicode, a preto pri ďalšom otvorení súboru iným prehliadačom dochádza opäť k nesprávnej reprezentácii znaku. Táto oprava však nedokáže pokryť všetky znaky. Príkladom je napríklad písmeno „ď“, ktoré má hodnotu GID 239 a jeho Unicode kód je U+010F. Z tohto dôvodu teda nestačí na opravu prevod z decimálnej hodnoty do hexadecimálnej.

Vzťah medzi GID a CID nie je v analyzovaných vzorkách pevne daný. Je možné ho však odvodiť vďaka poľu Differences, ktoré bolo popísané v rámci kapitoly 3.4. Základným postupom pri oprave takto poškodených súborov je nájdenie vzťahu medzu GID a CID pomocou tabuľky Differences, správna reprezentácia podľa GID do Unicode a vytvorenie objektu ToUnicode s týmito znakmi.

EBPFHN+Arial075
DNFMKG+Arial075
EJEIBP+Arial075
CKFHNJ+Arial075
EEBMAF+Arial075
ELMKCD+Arial075
EFOMMO+Arial075
EHEHAE+Arial075
EPCJBB+Arial075
EMDJIG+Arial075
EBCOLE+Arial075
ECOBAB+Arial075

Obr. 4.3: Zoznam objektov fontov Arial075

V rámci vybraných vzoriek nie je každý font definovaný iba jedným objektom, ktorý by sa zhodoval na každej strane dokumentu, kde je font použitý. Napríklad v rámci tretieho vydania Praktická elektronika z roku 2001 existuje 12 objektov pre font Arial075. Zoznam týchto objektov je možné vidieť na obrázku 4.3, kde je ich výpis vygenerovaný programom Infix PDF editor. Tieto fonty sa nijako nelíšia veľkosťou, šírkou alebo inými parametrami glyfov. Rozdiel medzi fontmi je iba v znakoch, ktoré obsahujú. Navyše každá strana obsahuje tento objekt a neodkazuje sa naň iba pomocou nepriameho objektu. Pre každý font je teda potrebné vygenerovať a vložiť doň vlastný objekt ToUnicode.

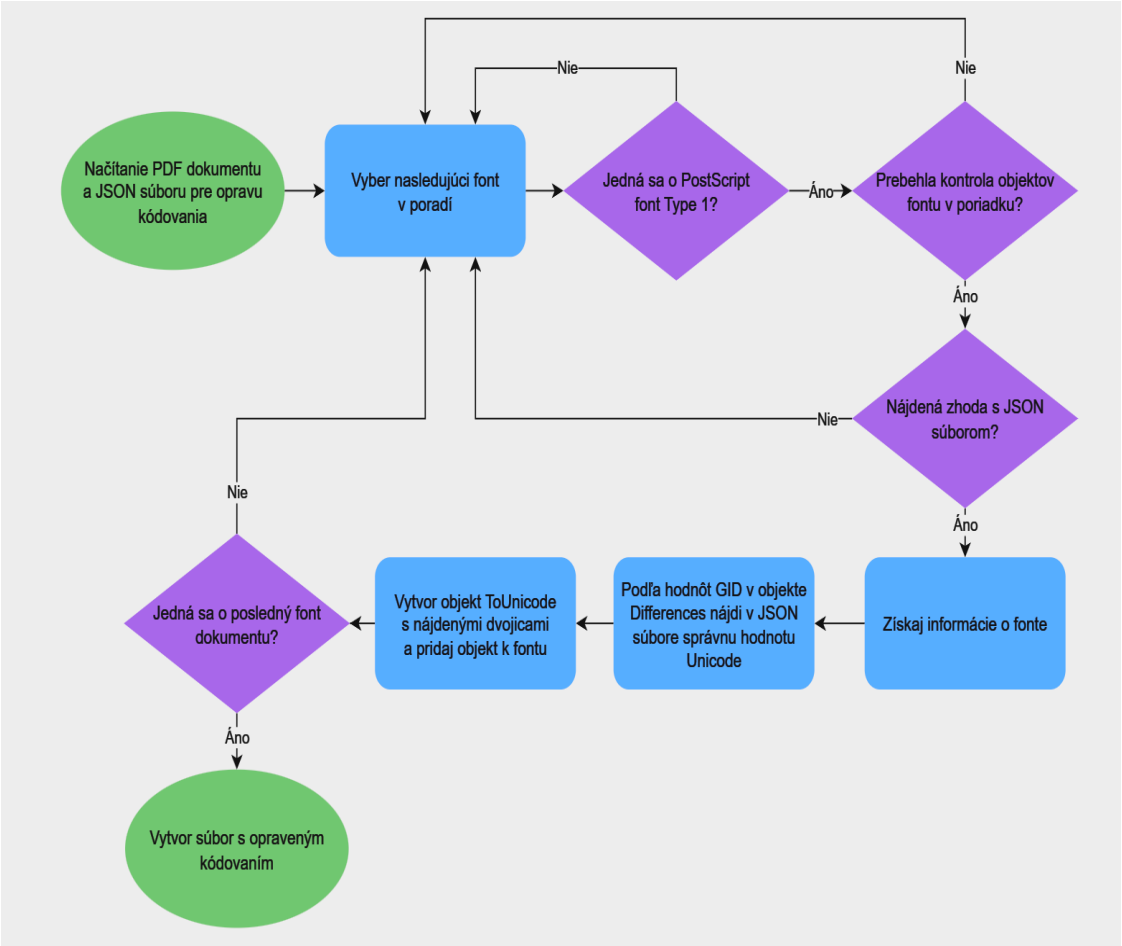
Na generovanie objektu ToUnicode bol vytvorený skript v jazyku Python. Python bol vybraný kvôli jednoduchej implementácii a jeho širokej kompatibilite. Kompletný zdrojový kód skriptu pre opravu s okomentovaním je zahrnutý v prílohách diplomovej práce.

4.3 Skript pre opravu kódovania

Pre opravu kódovania bol v rámci diplomovej práce vytvorený Python skript s názvom Type1toUnicode. Práca popisuje fungovanie skriptu verzie 0.4.0.

Pre potreby správnej funkčnosti skriptu boli nainštalované Python knižnice v nasledujúcich verziách:

- **colorama** - 0.4.6 - knižnica pre správne zobrazenie farebného textu v príkazovom riadku.
- **jellyfish** - 1.0.3 - knižnica pre výpočet Jaro Winklerovej vzdialenosti textových reťazcov.
- **Levenshtein** - 0.25.1 - knižnica pre výpočet Levenshteinovej vzdialenosti.
- **pypdf** - 4.2.0 - knižnica pre prácu s PDF súbormi (načítanie a zápis súboru).



Obr. 4.4: Vývojový diagram skriptu pre opravu súboru

Okrem inštalovaných knižníc skript využíva aj knižnice, ktoré sú súčasťou inštalácie Pythonu. Ide o knižnice:

- **argparse** - knižnica pre prácu s argumentmi, ktoré užívateľ zadáva v príkazovom riadku.
- **datetime** - knižnica pre prácu s časovou značkou, ktorú skript ukladá do metadát nového PDF súboru.
- **json** - knižnica pre načítanie a overenie správnej syntaxe súboru JSON s opravným mapovaním znakov.
- **logging** - knižnica pre prácu s informáciami, ktoré skript behom svojho priebehu zaznamenáva užívateľovi do textového súboru.
- **os** - knižnica pre interakciu s operačným systémom (vytvára nové požadované adresáre a overuje existenciu súborov).
- **sys** - knižnica použitá pre ukončenie skriptu v prípade chyby s príslušným chybovým kódom.

Zo všetkých uvedených knižníc boli importované iba nutne potrebné funkcie a triedy pre chod skriptu. Zo skriptu bol v závere vytvorený spustiteľný súbor, ktorého veľkosť sa týmto znižuje len na nevyhnutnú. Importovanie knižníc a ich častí je uvedené vo výpise 4.5.

Pre chybový výpis, ktorý sa zobrazuje užívateľovi do príkazového riadku po nesprávnom zadaní argumentov, je potrebné upraviť metódu error z triedy ArgumentParser. Úprava zabezpečuje pri zlom vstupe vypísanie názvu a verzie opravného skriptu. Následne metóda vypíše pomocnú hlášku pre prácu so skriptom, kde sú uvedené všetky argumenty a definovanie ich využitia a či je povinné dané argumenty pre spustenie zadať alebo nie. Po tomto výpise sa skript ukončí s chybovým stavom 1.

Výpis 4.5: Importovanie potrebných knižníc pre opravný skript

```
1 import logging, os, datetime
2 from json import load, JSONDecodeError
3 from sys import exit
4 from jellyfish import jaro_winkler_similarity
5 from Levenshtein import ratio as lev_ratio
6 from argparse import ArgumentParser
7 from pypdf import PdfReader, PdfWriter
8 from pypdf.generic import NameObject, StreamObject
9 from colorama import init
```

Po importovaní knižníc boli v skripte vytvorené štyri globálne konštanty. V konštante NAME je uvedený názov skriptu, ktorý sa zobrazuje v informáciách pre užívateľa. Verzia skriptu je taktiež pre tieto potreby uložená v konštante VERSION. Konštanta SUB_TYPE sa používa na definovanie typov fontov, ktoré skript opravuje. Skript sa zameriava iba na fonty typu Type1, aby sa typ nemohol v priebehu skriptu meniť, bol jeho názov priradený konštante.

4.3.1 Šablóna pre vytvorenie objektu ToUnicode

Ako je možné vidieť vo výpise 4.6, šablóna pre vytvorenie objektu je uložená v premennej TEMPLATE. Objekt je tvorený mapou znakov CMap v jazyku PostScript. Ide o štruktúru používanú na mapovanie sekvencií bajtov na znaky alebo glyfy.

Proces vytvárania mapy znakov je inicializovaný pomocou /CIDInit, čo je kľúčový identifikátor pre CID systém, ktorý sa používa na mapovanie znakov. Systémové informácie o mape sú definované v rámci /CIDSystemInfo. Definuje doplnok, radenie a register mapy. Register je vytváraný pomocou premennej name, ktorá bola pre potreby jedinečnosti vytvorená z kľúčového slova PAGE, ktoré je nasledované

číslom strany, na ktorej sa font, pre ktorý je mapa určená, nachádza. To však k jednoznačnosti nepostačuje, nakoľko sa na jednej strane nachádza viacero fontov. Preto bol za číslo strany pridaný znak plus a jedinečný názov fontu pre danú stranu (názov je prevzatý z objektu fontu Name). Rovnakým názvom je definovaný aj samotný názov mapy /CMapName.

Výpis 4.6: Šablóna objektu ToUnicode

```
1  TEMPLATE = \
2  """
3  /CIDInit /ProcSet findresource begin 12 dict begin begincmap
4  /CIDSystemInfo <<
5  /Registry ({name}+0) /Ordering (T1UV) /Supplement 0 >> def
6  /CMapName /{name}+0 def
7  /CMapType 2 def
8  1 begincodespacerange <{fchar_hex}> <{lchar_hex}>
9  endcodespacerange {lchar_dec} beginbfchar
10 {mapping}
11 endbfchar
12 endcmap CMapName currentdict /CMap defineresource pop end end
13 """
```

/CMapType definuje typ mapy, kde hodnota 2 znamená, že ide o mapovanie znakov podľa CID. Po definovaní typu je potrebné definovať rozsah kódového priestoru. Rozsah musí byť definovaný hexadecimálnymi číslami. Je udávaný pomocou premenných fchar_hex a lchar_hex. Ide o hodnoty objektu fontu FirstChar a LastChar prevedené do hexadecimálnej sústavy. Pokiaľ by rozsah presne nesedel s obsahom mapy, bola by nesprávne interpretovaná a vo výsledku by jej pridanie do súboru neopravilo žiadne kódovanie. Nemôže sa tu preto pevne stanoviť rozsah od minimálnej po maximálnu hodnotu.

V nasledujúcej časti mapy sa už udávajú konkrétne mapovania znakov. Premenná lchar_dec obsahuje dekadickú hodnotu LastChar a vyjadruje počet mapovaní, ktoré bude mapa obsahovať. Vo vzorových súboroch je pri kompletne rozbitých fontoch väčšinou dekadická hodnota FirstChar 1 a LastChar udáva počet znakov použitých vo fonte. Konkrétne mapovanie je obsiahnuté v premennej mapping, ktorej generovanie je vysvetlené v ďalšej časti diplomovej práce.

Posledná časť uzatvára definíciu mapy, ukladá ju ako zdroj a ukončuje prácu so slovníkom a CMap.

4.3.2 Štruktúra JSON pre opravu fontu

Opravné hodnoty pre glyfy jednotlivých fontov boli uložené do JSON štruktúry s názvom ToUnicode map. Na začiatku JSON súboru sa definuje jeho názov, verzia a dátum vytvorenia kvôli prehľadnosti. Po definícii týchto informácii bolo v JSON vytvorené pole s názvom fonts. Toto pole obsahuje všetky fonty, ktoré boli v rámci diplomovej práce analyzované a opravované.

Každý font v tejto štruktúre musí pre správnu funkčnosť obsahovať údaje name a data. Name definuje názov rodiny fontu (napríklad Arial). Toto pole sa využíva pri hľadaní správneho mapovania ešte neopraveného fontu. Meno analyzovaného fontu z PDF súboru je porovnávané s týmto poľom, a pokiaľ sú parametre podobnosti dostatočne vysoké, používa sa pre opravu vybratý font. V PDF súboroch obsahujú názvy fontov okrem presného názvu rodiny fontu napríklad aj šesť ďalších jedinečných znakov na začiatku názvu pre rozlíšenie jednotlivých fontov. Okrem týchto šiestich znakov potom môžu obsahovať aj rôzne číselné údaje popisujúce ich veľkosť alebo slovné údaje, ktoré popisujú, či sa napríklad jedná o kurzívu alebo tučné písmo. Z tohto dôvodu sa meno fontu v PDF súbore nemôže mapovať pomocou presnej zhody s poľom name v JSON štruktúre. Takéto porovnávanie by viedlo k veľkému množstvu duplicitných záznamov, nakoľko mapovanie pre celé rodiny fontov je rovnaké a nezáleží na veľkosti alebo vlastnostiach fontu. Presný proces tohto porovnávania aj s podmienkami výberu je popísaný v kapitole 4.3.3.

Niektoré rôzne rodiny fontov však môžu mať rovnaké opravné kódovanie glyfov. Pre zamedzenie takýmto duplicitám môže v JSON súbore font obsahovať aj pole s názvom alternativeNames. Toto pole nie je povinné pre funkčnosť opravného skriptu a iba zjednodušuje celú JSON štruktúru. Pokiaľ teda vyhľadávacia metóda nájde najlepšiu zhodu v zázname poľa alternativeNames, použije mapovanie fontu, ktorý pole s touto hodnotou obsahuje.

V objekte data sú uložené kľúče, ktoré reprezentujú mapovanie jednotlivých glyfov na hodnoty Unicode. Názvy hodnôt kľúčov sú citlivé na veľké a malé písmená, a teda napríklad mapovanie glyfu g50 môže byť iné ako mapovanie glyfu G50. V rámci analyzovaných PDF súborov takáto situácia nastala niekoľkokrát, a tak pri prípadnom rozširovaní štruktúry v ďalších verziách je potrebné túto skutočnosť brať do úvahy. Názov kľúču môže byť akýkoľvek reťazec, a teda nemusí označenie glyfov začínať písmenom g. Hodnota Unicode je vždy uvádzaná v hexadecimálnom tvare a mala by obsahovať štyri znaky. Použitie kratšieho alebo dlhšieho záznamu neovplyvní správnosť záznamu, ale pre jednotné zobrazenie boli pred kratšie záznamy doplnené nuly, ktoré dĺžku záznamu upravujú. Ukážku štruktúry JSON súboru je možné vidieť vo výpise 4.7.

Výpis 4.7: Vzor dát v JSON štruktúre pre opravu súborov

```
{
  "product": "ToUnicode␣map",
  "version": 0.7,
  "releaseDate": "2024-05-13T00:00:00.000Z",
  "fonts":[
    {
      "name": "Arial",
      "alternativeNames": ["Helvetica","Verdana"],
      "data": {
        "bar": "007C",
        "G56": "0038",
        "G57": "0039"
      }
    },{
      "name": "Times",
      "data": {
        "space": "0020",
        "G116": "0074",
        "G117": "0075",
        "G118": "0076"
      }
    }
  ]
}
```

Tab. 4.3: Prehľad rodín fontov s rovnakým opravným kódovaním

Názov fontu	**Alternatívne názvy**
Arial	Bookman, Courier, Forte, Franklin, Helvetica, +IBMPCDOS0, ItcEras, ItcErasBlackTT, Lucida, Switzerland, Tahoma, Verdana
Wingdings	
Webdings	
Times	
Symbol	
MSTT31	
Microsoft	Microsoft.Sans.Serif

V rámci diplomovej práce bolo do JSON štruktúry zaznamenaných 628 kľúčov s opravným kódovaním na Unicode. Išlo celkovo o 18 rôznych rodín fontov. Zá-

kladná štruktúra JSON súboru a opravné kódovanie boli vytvorené autorom práce. Vedúcim práce bola správnosť opravného kódovania overená a prípadne doladená. Vedúci práce taktiež vytvoril opravné mapovania pre niektoré rodiny fontov, napríklad Symbol, Wingdings alebo Webdings.Prehľad fontov, ktoré zdieľajú rovnaké opravné kódovanie, popisuje tabuľka 4.3. V tabuľke sú zobrazené názvy objektov fontov v JSON štruktúre a hodnoty poľa alternativeNames k ním priradeného.

4.3.3 Trieda UnicodeMapper

Metóda pre výber správneho mapovania fontu

Metóda find_similar_font, ktorá je súčasťou triedy UnicodeMapper, slúži k nájdeniu správneho fontu v popísanej JSON štruktúre. Vstupné parametre tejto metódy sú:

- **font_dict** - slovník, ktorý obsahuje dvojicu názvu primárneho fontu z JSON súboru a názvu alternatívneho fontu, ktorý je k nemu priradený.
- **search_font** - názov fontu, ktorý je a analyzovaný a ku ktorého názvu sa vyhľadáva správna oprava kódovania.

Metóda pracuje s tromi premennými, ktoré sa definujú na jej začiatku. Tieto premenné sú:

- **similarity** - miera podobnosti dvoch textových reťazcov vyjadrená v percentách. Jej hodnota je pri definovaní nula.
- **return_font** - primárny názov fontu z JSON súboru, ktorý bol použitý pri oprave. Pri definovaní je jej počiatočná hodnota nastavená na None.
- **f_name** - alternatívny názov fontu z JSON súboru. Pri definovaní je jej počiatočná hodnota nastavená na None.

Metóda po definovaní potrebných premenných prechádza cez všetky záznamy v slovníku font_dict. Následne vypočítava medzi nimi a názvom analyzovaného fontu Jaro-Winklerovu vzdialenosť, prípadne Levenshteinovu vzdialenosť.

Jaro-Winklerova vzdialenosť

Jaro-Winklerova vzdialenosť je metrika na porovnávanie podobnosti dvoch textových reťazcov. Metrika kladie dôraz na rovnaké začiatky textových reťazcov. Vyššia hodnota metriky znamená vyššiu podobnosť porovnávaných reťazcov a môže dosahovať hodnoty z intervalu 0 až 1.

Základom výpočtu je Jaro vzdialenosť, ktorá sa vypočítava na základe zhodných znakov a transpozície. Znaky sú považované za zhodné, ak sú rovnaké a nachádzajú sa od seba v určitej maximálnej vzdialenosti. Maximálna vzdialenosť je definovaná

ako polovica dĺžky kratšieho z porovnávaných reťazcov zaokrúhlená nadol. Transpozícia je počet rozdielnych pozícií medzi rovnakými znakmi, čo znamená, že znaky sa zhodujú, ale nie sú na rovnakých pozíciách v reťazcoch.[25]

Vzorec pre výpočet Jaro vzdialenosti je nasledovný:

$$J = \frac{1}{3}\left(\frac{m}{|s1|} + \frac{m}{|s2|} + \frac{m-t}{m}\right) \tag{4.1}$$

- **m** - počet zhodných znakov.
- **t** - polovica počtu rozdielnych pozícií medzi rovnakými znakmi.
- **|s1|, |s2|** - dĺžky porovnávaných reťazcov.[25]

Jaro-Winklerova vzdialenosť pridáva k Jaro vzdialenosti ďalší parameter, ktorý zvýrazňuje podobnosť na základe spoločného prefixu na začiatku reťazcov.[26]

Vzorec pre výpočet Jaro-Winklerovej vzdialenosti je:

$$JW = J + (l \times p\,(1 - J)) \tag{4.2}$$

- **J** - Jaro vzdialenosť.
- **l** - počet prvých zhodných znakov (maximálne 4).
- **p** - faktor škálovania (typicky 0,1).[26]

Levenshteinova vzdialenosť

Tiež je známa ako editačná vzdialenosť. Ide o metriku používanú na meranie rozdielov medzi dvomi textovými reťazcami. Udáva minimálny počet editačných operácií (vloženie, odstránenie alebo zmenu znaku), ktoré sú potrebné na premenu jedného reťazca na druhý.[27]

Levenshteinova vzdialenosť využíva tri základné editačné operácie:

- **vloženie** - pridanie jedného znaku do reťazca.
- **odstránenie** - odstránenie jedného znaku z reťazca.
- **zmenu** - zmena jedného znaku reťazca na iný znak. [27]

Pre výpočet vzdialenosti sa využívajú matice. Výpočet prebieha nasledovne:

1. Vytvorenie matice d o veľkosti (n + 1) x (m + 1), kde n a m sú dĺžky porovnávaných reťazcov.
2. Nastavenie d(i, 0) = i pre $0 \leq i \leq n$ a d(0, j) = j pre $0 \leq j \leq m$, reprezentujúce vzdialenosť od prázdneho reťazca.
3. Pre každý pár znakov i a j v oboch reťazcoch:
   - ak sú znaky zhodné ($a_i = b_j$), nastavia sa náklady na zmenu na 0
   - ak znaky nie sú zhodné, náklady na zmenu sa nastavia na 1
4. Vypočítanie hodnoty d(i, j) ako minimum z troch možných hodnôt:
   - Vloženie: d(i, j-1) + 1
   - Odstránenie: d(i-1, j) + 1

- Zmena: d(i-1, j-1) + náklady na zmenu

5. Výsledná vzdialenosť je hodnota v pravej dolnej bunke matice d(n, m). [27]

Metóda pre nájdenie názvu fontu

Metóda ukladá výpočet Jaro-Winklerovej vzdialenosti do premennej jw_sim. Hodnotu funkcie násobí ešte číslom 100. Pokiaľ je v danej iterácii hodnota vyššia ako doposiaľ najvyššia hodnota z predchádzajúcich iterácií a zároveň je hodnota vyššia ako 70, funkcia prepíše aktuálnu najvyššiu hodnotu podobnosti v premennej similarity a uloží názvy primárneho a alternatívneho fontu, ktoré zvolila z JSON súboru. Tento postup zabezpečuje takmer optimálny výber z JSON súboru.

V niektorých prípadoch, pokiaľ je názov fontu príliš dlhý, môže funkcia nenájsť správne mapovanie. Typicky ide o fonty, ktorých názov obsahuje informácie o použitej kurzíve, tučnom písme a veľkosti písma zároveň. Pre tieto prípady metóda ešte overuje druhú podmienku, ktorá môže výber zmeniť. Pokiaľ je teda hodnota Jaro-Winklerovej vzdialenosti menšia než 45 a zároveň je hodnota Levenshteinovej podobnosti vynásobenej číslom 100 vyššia ako 40, zvolí mapovanie, ktoré tejto podmienke odpovedá a prepíše hodnoty premenných similarity, f_name, return_font. Zdrojový kód metódy je uvedený vo výpise 4.8. Pokiaľ by v priebehu iterácií nenastala zhoda so žiadnym fontom v JSON súbore, metóda vracia hodnoty None. Tieto hodnoty následne v skripte rozhodujú o jeho ďalšom postupe.

Výpis 4.8: Metóda pre nájdenie názvu fontu

```
1  @classmethod
2  def find_similar_font(cls, font_dict, search_font):
3      similarity score and the corresponding font names
4      similarity, return_font, f_name = 0, None, None
5      for font_name, mapped_name in font_dict.items():
6          jw_sim = jaro_winkler_similarity(font_name,
7          search_font) * 100
8          if (jw_sim > similarity) and jw_sim >= 70:
9              similarity, f_name, return_font = jw_sim,
10             font_name, mapped_name
11         if (jw_sim<45 and
12         (lev_ratio(font_name,search_font)*100)>40):
13             similarity, f_name, return_font = jw_sim,
14             font_name, mapped_name
15     return f_name, return_font
```

Metóda pre získanie hodnoty Unicode

Metóda get_unicode_value slúži na získanie Unicode hodnoty pre konkrétny znak z fontu v rámci štruktúry dát. Parametre metódy sú nasledovné:

- **data** - dáta získané z JSON súboru.
- **font_name** - primárne meno fontu získané z metódy find_similar_font.
- **property_name** - označenie glyfu, ktorého mapovanie funkcia v JSON hľadá.

Vyhľadávanie iteruje cez každý objekt fonts v dátach JSON. Pokiaľ by objekt fonts nebol prítomný, funkcia vráti prázdny zoznam. Pokiaľ sa počas niektorej z iterácií nájde zhoda so zadaným fontom z parametra metódy, pokúsi sa získať údaje data z aktuálneho fontu a potom nájsť príslušnú hodnotu pre property_name. Ak záznam glyfu neexistuje v týchto údajoch, metóda vracia hodnotu None. Hodnotu None vráti takisto, pokiaľ žiadny zhodný font nie je nájdený. Zdrojový kód metódy je uvedený vo výpise 4.9.

Výpis 4.9: Metóda pre získanie hodnoty Unicode

```
1 @classmethod
2 def get_unicode_value(cls, data, font_name, property_name):
3     for font in data.get('fonts', []):
4         if font.get('name') == font_name:
5             return font.get('data',{}).get(property_name,None)
6     return None
```

4.3.4 Trieda File

Metóda pre overenie existencie súboru

Slúži na overenie existencie súboru a jeho prípony. Okrem toho, pokiaľ ide o súbor JSON, pokúša sa ho načítať a overiť jeho syntaktickú správnosť. Ak nastanú akékoľvek problémy, metóda vypíše chybové hlásenie a ukončí skript s príslušných chybovým kódom.

Metóda pracuje s dvomi vstupnými parametrami:

- **filename** - názov súboru, ktorý sa má overiť.
- **extension** - očakávaná prípona overovaného súboru.

Pomocou funkcie exists z knižnice os najskôr metóda overuje, či zadaný súbor existuje. Pokiaľ súbor neexistuje, do príkazového riadku vypíše chybové hlásenie v červenej farbe a ukončí skript s kódom 2.

Pokiaľ súbor existuje, získa jeho príponu pomocou funkcie splitext z knižnice os. Ak prípona nezodpovedá očakávanej prípone (bez ohľadu na veľkosť písmen v prípone), vypíše chybové hlásenie v červenej farbe a ukončí skript s kódom 3.

Ak je prípona súboru .json, pokúsi sa metóda súbor otvoriť a načítať jeho obsah pomocou funkcie load z knižnice json.

Chyba pri dekódovaní JSON súboru (napríklad syntaktická chyba v súbore) vypíše červenou farbou chybové hlásenie a ukončí skript s kódom 4. V rámci chybového výpisu sa zobrazuje aj presný riadok, kde v JSON súbore došlo k chybe. Pri akejkoľvek inej výnimke sa táto výnimka vypíše do príkazového riadku červenou farbou a skript sa ukončí s kódom 5. Všetky chybové kódy a ich vysvetlenie, ktoré môžu v skripte nastať, sú uvedené v tabuľke 4.4.

Výpis 4.10: Metóda pre overenie existencie súboru

```
1  @classmethod
2  def validate(cls, filename, extension):
3      try:
4          if not os.path.exists(filename):
5              print(f"\033[31mFile␣does␣not␣exist:
6  ␣␣␣␣␣␣␣␣␣␣␣␣{filename}\033[0m")
7              exit(2)
8          _, file_extension = os.path.splitext(filename)
9          if file_extension.lower() != extension.lower():
10             print(f"\033[31mFile␣is␣not␣in␣expected␣format.
11 ␣␣␣␣␣␣␣␣␣␣␣␣Expected␣{extension}\033[0m")
12             exit(3)
13         if file_extension.lower() == '.json':
14             with open(filename,'r',encoding='utf-8')
15             as json_file:
16                 load(json_figle)
17     except JSONDecodeError as e:
18         print(f"\033[31mError␣in␣parsing␣JSON␣file:
19 ␣␣␣␣␣␣␣␣{filename}\n{e}\033[0m")
20         exit(4)
21     except Exception as e:
22         print(f"\033[31mAn␣exception␣occurred:␣{e}\033[0m")
23         exit(5)
```

Metóda pre načítanie JSON súboru

Metóda load_json sa využíva na otvorenie zadaného JSON súboru s daným kódovaním a vracia jeho obsah. Pre jej funkčnosť sú požadované dva parametre:

- **filename** - názov súboru, ktorý sa má načítať.
- **encoding** - kódovanie, v ktorom sa má súbor otvoriť. Predvolená hodnota je nastavená na kódovanie UTF-8.

Tab. 4.4: Prehľad chybových kódov programu

Chybový kód	Význam chybového kódu
1	Chybne zadané argumenty pri spúšťaní skriptu
2	Zadaný súbor neexistuje
3	Nesprávna prípona zadaného súboru
4	Chyba pri dekódovaní JSON súboru
5	Všeobecná chyba pri overovaní súboru

Pomocou funkcie open sa zadaný súbor otvára v režime na čítanie, čo definuje parameter r tejto funkcie. Súbor sa otvára ako json_file. Následne funkcia vracia obsah súboru pomocou funkcie load z knižnice json, ktorá konvertuje JSON formát na zodpovedajúcu dátovú štruktúru. Zdrojový kód metódy je uvedený vo výpise 4.11.

Výpis 4.11: Metóda pre načítanie JSON súboru

```
1  @classmethod
2  def load_json(cls, filename, encoding='utf-8'):
3      with open(filename, 'r', encoding=encoding) as json_file:
4          return load(json_file)
```

Metóda pre aktualizáciu metadát

Výpis 4.12: Metóda pre aktualizáciu metadát

```
1  @classmethod
2  def update_metadata(cls, existing_metadata):
3      now = datetime.datetime.now()
4      formatted_time = 'D:' + now.strftime('%Y%m%d%H%M%S')
5      new_metadata = {
6          '/ModDate': formatted_time,
7          '/Producer': NAME + "␣" + VERSION
8      }
9      updated_metadata = existing_metadata.copy()
10     updated_metadata.update(new_metadata)
11     return updated_metadata
```

Do funkcie update_metadata sú predávané existujúce metadáta opravovaného PDF súboru. Metadáta sú predávané ako parameter existing_metadata.

Na začiatku získa metóda aktuálny dátum a čas, ktorý je uložený do premennej now. Týmto časom sa v metadátach upravuje dátum poslednej zmeny súboru. Čas je

potrebné naformátovať do správneho formátu, aby bol správne v PDF prehliadačoch interpretovaný. Hodnota musí začínať reťazcom „D:“, za ktorým nasledujú údaje o roku, mesiaci, dni, hodinách, minútach a sekundách. Do formátu je možné za údaje o sekundách pridať aj informácie o časovom pásme, v ktorom je tento čas interpretovaný.

Aktualizované hodnoty metadát sa ukladajú do slovníka new_metadata. Skript upravuje iba hodnoty programu, ktorý je označovaný ako producent, a času poslednej úpravy dokumentu. Ako producent sa udávajú názov skriptu a jeho verzia oddelené medzerou. Tieto hodnoty sú získané z konštánt na začiatku skriptu.

Do premennej updated_metadata skript vytvorí kópiu existujúcich metadát pomocou metódy copy, aby sa zachovali pôvodné metadáta v nezmenenej podobe. Následne aktualizuje kópiu existujúcich metadát slovníkom new_metadata pomocou metódy update, ktorá pridá alebo nahradí zodpovedajúce kľúče a hodnoty. Ostatné hodnoty pôvodných metadát teda v súbore ostávajú nezmenené.

4.3.5 Hlavná funkcia skriptu

Na začiatku hlavnej funkcie skriptu sa inicializuje knižnica colorama, ktorá slúži pre správne zobrazovanie farebného textu v príkazovom riadku, odkiaľ je skript volaný. Bez tejto inicializácie sa farebnosť správne neidentifikuje a do príkazového riadku sa výpisy zobrazujú chybne.

Okrem inicializácie knižnice colorama je potrebné vytvoriť aj premenné, ktoré zaznamenávajú štatistiku správnosti opráv jednotlivých fontov v opravovanom PDF súbore. Pre túto štatistiku sú vytvorené tri počítadlá:

- **cnt_skipped** - uchováva počet fontov, ktoré boli v priebehu opravy kompletne preskočené, a teda u nich nedošlo ku žiadnej oprave kódovania.
- **cnt_rep_part** - uchováva počet fontov, ktoré boli v priebehu opravy opravené len čiastočne. U niektorých glyfov v tomto prípade nebola nájdená zhoda s obsahom JSON štruktúry, a tak ich kódovanie nie je ani po oprave správne.
- **cnt_rep_comp** - uchováva počet fontov, ktorých kódovanie glyfov bolo skriptom počas jeho priebehu kompletne opravené, a teda takýto font považuje za kompletne opravený.

Argumenty príkazového riadku a logovanie

V hlavnej funkcii skriptu bol vytvorený objekt argparser, ktorý sa používa na spracovanie argumentov z príkazového riadku. Do tohto objektu boli následne pridané tri argumenty. Prvý argument bol pomenovaný pdf_file a slúži na definíciu dokumentu pre opravu kódovania, ide o povinný argument vyžadovaný pri spúšťaní skriptu. Je

možné ho v príkazovom riadku použiť pomocou prepínača -p alebo jeho dlhou verziou –pdf_file. Očakáva reťazec typu string, ktorý reprezentuje cestu k dokumentu. Argument sa využíva pre vytvorenie inštancie PdfReader, ktorým je špecifikovaný dokument načítaný. Pre umožnenie vykonávania zmien v dokumente bolo potrebné vytvoriť aj inštanciu PdfWriter, ktorá sa používa na zápis úprav. Argument obsahuje aj informačný výpis pre užívateľa, ktorý sa zobrazuje pri nesprávnom zadaní argumentov pri spúšťaní skriptu alebo pri spustení skriptu s argumentom -h (help), ktorý zobrazí výpis pomocných výpisov.

Druhým argumentom je font_map, ktorý je takisto typu string, a taktiež ide o povinný argument. Pomocou tohto argumentu je možné kódu definovať, ktorý JSON súbor s opravným kódovaním má použiť pre opravu. Je možné ho použiť prepínačom -f alebo jeho dlhou verziou –font_map. Argument skriptu sa používa pri volaniach metód pre overenie správnosti súborov a načítania obsahu z JSON súboru, ktoré boli opísané v predchádzajúcich častiach diplomovej práce. Taktiež obsahuje popis, ktorý sa zobrazuje ako pomocný výpis a v jednoduchosti popisuje využitie argumentu.

Tretí argument má názov verbose. Ide o nepovinný argument, ktorý signalizuje skriptu, či má používať štandardné alebo rozšírené logovanie. Argument je možné použiť prepínačom -v alebo jeho dlhou verziou –verbose. Predvolená hodnota argumentu je False a po jeho zadaní v príkazovom riadku sa jeho hodnota mení na True. Týmto spôsobom je v skripte možné jednoducho ovládať, či sa majú určité informácie zaznamenávať do logovacích súborov. Jeho funkcionalita je takisto opísaný v prípade zobrazenia pomocných výpisov skriptu.

Pomocou funkcie getLogger z knižnice logging skript vytvára objekt logger. Na identifikáciu logovacej inštancie slúži zvolený názov PdfRepair. Úroveň logovania sa nastavuje na hodnotu DEBUG, čo znamená, že logger bude zaznamenávať všetky správy s úrovňou DEBUG a vyššou (INFO, WARNING, ERROR a CRITICAL). Logy pre jednotlivé súbory sú zaznamenávané do textových súborov, ktorých názvy pozostávajú z názvu originálneho súboru a pridanej koncovky „_log“. Kvôli prehľadnosti sa všetky logy uchovávajú do priečinka Log. Pokiaľ takýto priečinok v mieste spustenia skriptu neexistuje, skript si ho vytvorí sám. Pri zápise do súboru sa logovací súbor otvára v režime zápisu, čo znamená, že pokiaľ obsahuje nejaké údaje, budú tieto údaje prepísané novými. Do súboru sa však skript nepokúsi zapisovať skôr, ale až pri prvom vyvolanom zápise do logu. Pokiaľ by teda neprebehol v priebehu skriptu žiadny zápis a existoval by textový súbor s nejakými údajmi, tieto údaje zostanú nezmenené aj po skončení priebehu skriptu.

Premenná formatter v skripte určuje formát logovacích správ. V tomto prípade sa bude zaznamenávať len samotná správa. Okrem správy by bolo možné pridať aj napríklad informácie o čase záznamu, čo však v tomto prípade nebolo požadované

```
Page "3" -> Font "/JGGABE+MSTT31bf7273360044017Iee" -> ToUnicode already exists -> skipping
Page "3" -> Font "/JGGADL+MSTT3184fd6d960045018ee" -> ToUnicode already exists -> skipping
Page "3" -> Font "/JGGAJK+MSTT31bf7273360035013Iee" -> ToUnicode already exists -> skipping
Page "4" -> Font "/Symbol" -> FontDescriptor entries missing -> skipping
Page "19" -> Font "/BMOJOI+Courier.New037.5" -> no matching font section found in JSON file -> using alternative name "Courier" from section: "Arial"
Page "33" -> Font "/Helvetica-Bold" -> table Differences does not exist -> skipping
Page "42" -> Font "/FBOFFC+Symbol" -> ToUnicode already exists -> skipping
File "myPDF.pdf", "142" fonts found, "10" fonts skipped, "0" fonts repaired partially, "132" fonts repaired completely
```

Obr. 4.5: Obsah súboru s logmi

a išlo by o nadbytočné informácie. Všetky logovacie správy sa zapisujú len do logovacieho súboru. Do príkazového riadku sa dostávajú iba výnimky, ktoré ukončia chod skriptu a štatistické správy o oprave fontov. Pokiaľ sa do logovacieho súboru zapíše nejaký záznam, je o tom užívateľ informovaný priamo do príkazového riadku s uvedenou cestou, kde sa daný súbor s logmi nachádza. V prípade správneho priebehu opravy sa štatistika opravy okrem výpisu do príkazového riadku zaznamená aj na koniec logovacieho súboru. Príkladný výpis logu je zobrazený na obrázku 4.5.

Do logovacieho súboru v prípade spustenia rozšíreného logovania skriptu môže byť vypísaných 10 rôznych informačných hlášok okrem štatistiky opravy. Tieto hlášky môžu informovať o:

1. nenájdení objektu fontu na strane - do logu sa zaznamená číslo strany, ktorá neobsahuje žiadne objekty fontov.
2. inom type fontu ako Type1 - do logu sa zaznamená číslo strany, na ktorej sa font nachádza, jeho názov (pokiaľ nejaký má) a o aký typ fontu ide.
3. chýbajúcej tabuľke rozdielov - zaznamenáva informáciu o chýbajúcom objekte Differences. Zároveň konkretizuje stranu súboru a názov fontu, u ktorého tento objekt nebol nájdený.
4. chýbajúcich informáciách v objekte FontDescriptor - v objekte sa nenachádzajú parametre, ktoré sú potrebné pre opravu kódovania. Zaznamenáva aj číslo strany a názov fontu.
5. chýbajúcich informáciách v tabuľke rozdielov - pokiaľ v tabuľke rozdielov chýbajú niektoré glyfy, ktoré font obsahuje. Zaznamenáva aj číslo strany a názov fontu.
6. existujúcom objekte ToUnicode - pokiaľ analyzovaný font už v sebe obsahuje objekt ToUnicode. Zaznamenáva číslo strany, názov fontu a informáciu o existencii objektu.
7. nenájdení podobnosti medzi názvom analyzovaného fontu a názvami fontov v JSON súbore. Okrem tejto informácie zaznamená aj číslo strany a názov fontu.
8. použití opravných údajov na základe alternatívneho mena v JSON súbore -

pokiaľ skript nenájde zhodu s primárnymi názvami fontov v JSON súbore, ale s alternatívnym, použije opravu kódovania podľa primárneho názvu. Do logu však zaznamená číslo strany, názov fontu, použitý alternatívny názov fontu a primárny názov fontu, ktorému tento alternatívny názov podlieha.

9. nenájdení opravného kódovania pre analyzovaný glyf v JSON štruktúre - pokiaľ v štruktúre nenájde zhodu s názvom glyfu a niektorého z kľúčov v JSON, zaznamená číslo strany, názov fontu a názov glyfu, ktorý nebol nájdený.
10. nevytvorení výstupného PDF súboru - pokiaľ skript vo svojom priebehu prejde všetkými fontmi PDF dokumentu, ale žiadny z nich nie je schopný opraviť, zaznamená pre užívateľa informáciu, že nebol vytvorený žiadny výstupný PDF súbor. Táto hláška sa okrem logu zobrazí aj v príkazovom riadku červenou farbou.

Spracovanie strán a fontov dokumentu

Skript po načítaní údajov z JSON súboru vytvára dvojice primárnych a alternatívnych názvov fontov, ktoré v JSON štruktúre našiel. Dvojice sú uložené v premennej font_dict, ktorá je typom slovník. Pre uloženie fontov do slovníka je potrebné prejsť všetky objekty fontov JSON súboru pomocou cyklu for. Prvá z dvojice hodnôt slovníka reprezentuje alternatívny názov fontu a druhá z dvojice reprezentuje primárny názov fontu. Pokiaľ skript pri prechode narazí na primárny názov fontu, vytvorí dvojicu s rovnakými hodnotami, a teda v oboch hodnotách je primárny názov fontu z objektu štruktúry. Kód vytvárajúci tento slovník je uvedený vo výpise 4.13.

Výpis 4.13: Naplnenie slovníku s názvami fontov

```
1 json_data = File.load_json(args.font_map)
2 font_dict = {}
3 for font in json_data['fonts']:
4     font_dict[font['name']] = font['name']
5     if 'alternativeNames' in font:
6         for alt_name in font['alternativeNames']:
7             font_dict[alt_name] = font['name']
```

Prechod skriptu všetkými stranami dokumentu bol zabezpečený cyklom for. Využíva na to atribút pages objektu reader, ktorý obsahuje zoznam strán v PDF dokumente. Pomocou funkcie enumerate boli vytvorené dvojice, pričom každá dvojica obsahuje index (premenná pagenum) a hodnotu (premenná page). Pagenum teda slúži ako počítadlo stránok a page je objekt reprezentujúci konkrétnu stranu v PDF dokumente.

Následne ďalším cyklom for skript iteruje cez všetky fonty použité na danej strane dokumentu. Prechádza cez všetky fonty v slovníku fontov (/Font), ktorý

je súčasťou objektu /Resources danej strany. Pomocou metódy items vráti dvojice kľúč a hodnota zo slovníka, kde font je kľúč (používa názov fontu) a data je hodnota (informácie o danom fonte).

V ďalšej časti sa skript zameriava na získanie informácií o fontoch. Do premennej _dataobj uloží objekt reprezentujúci font. Hodnota data bola získaná v predchádzajúcom cykle. Po načítaní objektu fontu skontroluje jeho typ a názov. Pokiaľ typ fontu nezodpovedá hodnote premennej SUB_NAME („Type1“), cyklus inkrementuje hodnotu počítadla pre preskočené fonty a preskočí na ďalšiu iteráciu, v ktorej začne spracovávať ďalší font bez zmeny predošlého fontu. V režime rozšíreného logovania taktiež zaznamená príslušnú informáciu o preskočení fontu do logu.

Pre efektívnejšie spracovanie objektov fontov si skript ukladá všetky názvy fontov, ktoré už boli spracované, do premennej analyzed_fonts . Táto premenná je definovaná ako množina. Pokiaľ sa teda názov fontu už nachádza v tejto množine, skript jeho spracovanie preskočí. Počítadlo preskočených fontov sa však neinkrementuje, pretože objekt fontu v súbore je vždy len jeden a inkrementovanie počítadla pri tomto kroku by malo za následok duplicitné započítanie fontu do štatistiky. Ak sa názov nezhoduje s obsahom množiny, názov fontu bude pridaný do množiny a analýza fontu bude pokračovať na ďalšie kroky.

Výpis 4.14: Kontrola objektov FontDescriptor a Differences

```
1  if '/FirstChar' not in _dataobj or '/LastChar' not in _dataobj
2      if args.verbose:
3          logger.debug('Page␣"%s"␣->␣Font␣"%s"␣->␣FontDescriptor
4  ␣␣␣␣␣␣␣␣entries␣missing␣->␣skipping', pagenum+1, _fontname)
5      cnt_skipped += 1
6      continue
7  fchar = _dataobj['/FirstChar']
8  lchar = _dataobj['/LastChar']
9  if ((lchar-fchar)+2) != len(_dataobj['/Encoding']
10 ['/Differences']):
11     if args.verbose:
12         logger.debug('Page␣"%s"␣->␣Font␣"%s"␣->␣no␣ToUnicode
13 ␣␣␣␣␣␣␣␣but␣Differences␣incomplete␣->␣skipping', pagenum+1,
14         _fontname)
15     cnt_skipped += 1
16     continue
```

Ďalšie z dôležitých kontrol analyzovaného fontu sú kontrola existencie objektu fontu Encoding a najmä kontrola existencie objektu Differences v ňom. Pokiaľ by objekt Differences neexistoval, znamená to, že v kódovaní nie je žiadna chyba a font netreba teda nijako opravovať. V prípade absencie objektu Differences teda skript

inkrementuje počítadlo preskočených fontov a pokračuje na ďalší font. Pokiaľ je spustený režim rozšíreného logovania, do logu sa ukladá hláška o tomto preskočení.

V objekte fontu patria medzi potrebné aj objekty FirstChar a LastChar. Tieto objekty vo fonte popisujú index prvého a posledného glyfu, ktoré font obsahuje. Z ich hodnôt je možné určiť počet glyfov v objekte fontu. Práve kvôli tejto možnosti je ich existencia kľúčová pre vytvorenie správnej opravnej tabuľky ToUnicode. Pokiaľ tieto hodnoty v objekte neexistujú, skript preskakuje font, inkrementuje počítadlo preskočených fontov a v prípade rozšíreného logovania zaznamenáva preskočenie. Ak tieto hodnoty existujú porovnáva sa ich rozdiel (LastChar - First Char + 2) s počtom záznamov v tabuľke rozdielov Differences. Týmto krokom sa overuje, či tabuľka rozdielov obsahuje záznamy o všetkých glyfoch vo fonte. Ak font nevyhovie tejto kontrole, je preskočený. Postup tejto kontroly je uvedený vo výpise 4.14.

Poslednou kontrolou, ktorá môže preskočiť celý font, je kontrola podobnosti názvu analyzovaného fontu a fontov v JSON. Pre túto kontrolu sa využívajú výstupy metódy find_similar_font z triedy UnicodeMapper, ktorá bola opísaná v kapitole 4.3.3. Výstupy metódy sa ukladajú do premenných alt_fontname a mapped_fontname. Pokiaľ skript nenájde dostatočnú podobnosť medzi názvami, preskočí analyzovaný font, inkrementuje počítadlo preskočených fontov, a pokiaľ je zvolené rozšírené logovanie, poznačí tieto informácie do logu. V prípade, že funkcia vyberie meno fontu z alternatívnych názvov v JSON súbore, skript vytvorí v logu záznam o tomto výbere. Tento záznam sa vytvára aj bez spustenia rozšíreného logovania. Kód, ktorý zabezpečuje túto kontrolu, je uvedený vo výpise 4.15.

Výpis 4.15: Hľadanie a overenie nájdenia fontu v JSON

```
1  alt_fontname, mapped_fontname = UnicodeMapper.find_similar
2  _font(font_dict, _fontname)
3  if mapped_fontname is None:
4      if args.verbose:
5          logger.debug('Page␣"%s"␣->␣Font␣"%s"␣->␣no␣matching
6  ␣␣␣␣␣␣␣␣mapping␣name␣found␣in␣JSON␣file␣->␣skipping',
7          pagenum+1, _fontname)
8      cnt_skipped += 1
9      continue
10 if mapped_fontname is not None and
11 mapped_fontname not in _fontname:
12     logger.debug('Page␣"%s"␣->␣Font␣"%s"␣->␣no␣matching␣font
13 ␣␣␣␣section␣found␣in␣JSON␣file␣->␣using␣alternative␣name␣"%s"
14 ␣␣␣␣from␣section:␣"%s"', pagenum+1, _fontname, alt_fontname,
15     mapped_fontname)
```

Vytvorenie mapovania znakov

Pre vytvorenie mapy bolo potrebné najskôr definovať premennú _mapping. Ide o prázdny zoznam, ktorý bude následne naplnený správnymi hodnotami potrebnými pre mapovanie. Ďalším cyklom for prechádza skript cez hodnoty objektu fontu Differences. Začína od druhého prvku objektu, pretože prvý prvok vždy obsahuje počiatočný index. Pokiaľ hodnota prvku nie je textový reťazec, upraví jej typ na reťazec. Takáto situácia nastáva najmä, pokiaľ je GID definované ako číselná hodnota. V niektorých analyzovaných ročníkoch sa vyskytovali glyfy práve s takýmto typom GID. Ak je GID zaznamenané v Differences ako textový reťazec, skript odstráni prvý znak tohto reťazca. Ide o znak „/“, ktorým začína reťazec s GID.

K indexu analyzovaného znaku sa pripočítava dekadická hodnota FirstChar a výsledok sa upravuje na minimálne dvojmiestny hexadecimálny zápis. Hodnota FirstChar musí byť pripočítaná k indexu kvôli možnému posunutému začiatku indexácie znakov vo fonte. Hexadecimálny zápis sa vyžaduje pre správnu funkčnosť mapovania v následne vytváranom objekte ToUnicode. Takto upravená hodnota indexu zodpovedá CID každého glyfu vo fonte.

Nájdenie správneho mapovania zabezpečuje metóda get_unicode_value z triedy UnicodeMapper, ktorá bola popísaná v kapitole 4.3.3. Výstup metódy sa ukladá do premennej unicode_value. Pokiaľ je hodnota premennej po skončení funkcie None, znamená to, že nebolo nájdené opravné mapovanie pre analyzovaný glyf. V prípade nenájdenia zhody sa vytvorí záznam do logu, ktorý o tomto výsledku informuje užívateľa. Skript však priradí k analyzovanému znaku mapovanie na medzeru. Tento mechanizmus zabezpečí, že vo výslednom objekte ToUnicode budú obsiahnuté všetky znaky, aj keď ich mapovanie nebolo opravené kompletne. Pokiaľ by sa v mape objektu nenachádzali všetky znaky, prehliadače PDF dokumentov by videli túto mapu ako chybnú a nebrali by jej existenciu do úvahy, čo by spôsobilo, že sa vo výslednom súbore neopraví ani kódovanie znakov, ktoré boli v priebehu analýzy správne opravené. Hodnota medzery v Unicode predstavuje hexadecimálnu hodnotu 0020. Vytvorený záznam sa teda pridáva do premennej _mapping. Okrem týchto krokov je ešte v prípade nenájdenia zhody nastavený ukazovateľ chyby error_unicode na hodnotu True. Po poslednej iterácii cyklu cez všetky hodnoty Differences skript vyhodnocuje podľa hodnoty tejto premennej, či bol celý font opravený správne alebo čiastočne. Na základe nej teda inkrementuje správne počítadlo počtu opravených alebo čiastočne opravených fontov a prípadne vráti hodnotu premennej error_unicode na False. Python kód, ktorý zabezpečuje vytvorenie popísaného mapovania, je vo výpise 4.16.

Výpis 4.16: Vytvorenie mapovania znakov

```
fchar_hex = f"{fchar:X}".zfill(2)
lchar_hex = f"{lchar:X}".zfill(2)
name = f"PAGE{pagenum+1}+{font[1:]}"
_mapping = []
for idx, char in enumerate(_dataobj['/Encoding']
['/Differences'][1:]):
    if not isinstance(char,str):
        char = str(char)
    else:
        char = char[1:]
    idx = (f"{(idx+fchar):X}".zfill(2))
    unicode_value = UnicodeMapper.get_unicode_value(json_data,
    mapped_fontname, char)
    if unicode_value is None:
        logger.debug('Page␣"%s"␣->␣Font␣"%s"␣->␣Glyph␣"%s"
␣␣␣␣␣␣␣␣not␣found␣in␣mapping', pagenum+1, _fontname, char)
        _mapping.append(f"<{idx}>␣<0020>")
        error_unicode = True
        continue
    _mapping.append(f"<{idx}>␣<{unicode_value}>")
if error_unicode is True:
    cnt_rep_part += 1
    error_unicode = False
else:
    cnt_rep_comp +=1
```

4.3.6 Vytvorenie a priradnie objektu ToUnicode k fontu

Po prejdení všetkých glyfov vo fonte skript premení zoznam reťazcov v _mapping na jediný reťazec, v ktorom je každý prvok zoznamu oddelený novým riadkom. Skript vytvára novú premennú _stream_data, ktorá bude použitá na uloženie dát mapovania. Jej obsah je vytvorený pomocou šablóny pre ToUnicode a údajov získaných z práve opravovaného fontu. Medzi tieto údaje patria:

- **name** - názov vytvorenej mapy. Názov pozostáva z čísla strany, na ktorom sa font prvýkrát v dokumente vyskytuje, a názvu opravovaného fontu.
- **fchar_hex** - hexadecimálna hodnota objektu FirstChar.
- **lchar_hex** - hexadecimálna hodnota objektu LastChar
- **lchar_dec** - premenná definuje počet záznamov v opravnom mapovaní v dekadickej hodnote. Vypočítava sa ako rozdiel dekadických hodnôt LastChar

a FirstChar. K rozdielu je nutné pripočítať ešte hodnotu 1.

- **lchar_hex** - vytvorené opravné mapovanie. Skript sem odovzdáva výslednú hodnotu premennej _mapping.

Obsah šablóny je kódovaný do formátu UTF-8. Použitím Flate kompresie je obsah premennej komprimovaný, ide o bežný spôsob kompresie v PDF dokumentoch.

Objekt fontu je následne aktualizovaný a je pridaný nový objekt ToUnicode. Pridanie ToUnicode teraz zabezpečí možnosť vyhľadávania a kopírovania obsahu v dokumente.

Vytvorenie výsledného PDF súboru

V záverečnej časti skript overuje, či v jeho priebehu došlo k oprave nejakého fontu. Tieto informácie mu zabezpečujú hodnoty uložené v počítadlách. Pokiaľ má aspoň jedno z počítadiel opravených fontov hodnotu inú než nula, skript vytvorí nový opravený PDF súbor v rovnakom umiestnení, odkiaľ je skript spúšťaný. Z názvu pôvodného súboru odstráni príponu a nahradí ju reťazcom „_repaired.pdf“. Predpokladá sa, že súbor obsahuje v názve len jednu bodku a tou je prípona súboru. Takto upravený názov je uložený do premennej outfile.

Pomocou cyklu for prejde cez všetky strany v premennej reader a pridá ich do premennej writer. Premenná teraz obsahuje všetky strany pôvodného dokumentu aj s opravenými fontmi. Následne vkladá do premennej writer ešte aktualizované metadáta z pôvodného súboru. Aktualizované metadáta vytvorí metóda update_metadata z triedy File. Po pridaní metadát sa súbor zapisuje na disk pod názvom z obsahu premennej outfile.

```
PS C:\Users\Admin\Desktop\Dip> python Type1toUnicode.py -p _PE10_2002.pdf -f to_unicode.json -v
Some font(s) have undefined character(s) mapping, please see log file _PE10_2002_log.txt in Log directory.
File _PE10_2002.pdf, 142 fonts found, 10 fonts skipped, 1 fonts repaired partially, 131 fonts repaired completely
```

Obr. 4.6: Výpis do príkazového riadku po skončení skriptu

V prípade spustenia skriptu s rozšíreným logovaním je do logu poznamenaná štatistika opravy fontov v súbore. Štatistika obsahuje celkový počet analyzovaných fontov, počet preskočených fontov, počet čiastočne opravených fontov a počet kompletne opravených fontov. Pokiaľ v priebehu skriptu dôjde k oprave niektorých fontov len čiastočne, skript vypíše pre užívateľa do príkazového riadku žltou farbou upozornenie o tejto skutočnosti s odkazom na umiestnenie logového súboru, kde nájde ďalšie informácie. Pokiaľ by nedošlo v priebehu skriptu k oprave žiadneho fontu,

skript nevytvára žiadny nový PDF súbor a informuje užívateľa, že táto situácia nastala. Táto situácia môže nastať najmä, pokiaľ by skript nedokázal ani čiastočne opraviť žiadny z fontov analyzovaného PDF súboru.

Štatistika opravy sa vždy po úspešnom skončení skriptu vypisuje užívateľovi do príkazového riadku zelenou farbou. Príklad spustenia skriptu a jeho výpisy do príkazového riadku sú zobrazené na obrázku 4.6.

5 Kompilácia a zverejnenie skriptov

Oba skripty vytvorené v rámci diplomovej práce boli skompilované do spustiteľného súboru (.exe). Kompilácia Python skriptu do spustiteľného súboru zjednodušuje distribúciu a používanie, zvyšuje bezpečnosť a ochranu kódu, zlepšuje výkon a kompatibilitu. Tieto výhody môžu byť obzvlášť dôležité pri vývoji aplikácií určených pre širšiu verejnosť alebo komerčné použitie. Vysvetlenie jednotlivých výhod:

1. Jednoduchšia distribúcia a inštalácia - používatelia nemusia mať nainštalovaný Python a spustiteľný súbor môže obsahovať všetky potrebné závislosti a knižnice, čo znamená, že používatelia nemusia manuálne inštalovať žiadne doplnkové balíčky.
2. Bezpečnosť a ochrana kódu - kompiláciou sa znižuje riziko, že niekto bude môcť ľahko zmeniť zdrojový kód. Aj keď úplná dekompilácia je stále možná, zvyšuje to úroveň ochrany kódu. Používatelia nemôžu omylom upraviť alebo poškodiť skript, čo znižuje riziko chybných úprav a následných problémov.
3. Výkon a optimalizácia - v niektorých prípadoch môže byť .exe súbor optimalizovaný na rýchlejšie spustenie ako interpretovaný skript. Pre určité úlohy môže byť kompilovaný kód vykonávaný efektívnejšie ako interpretovaný kód.
4. Kompatibilita - súbor môže byť spustený v akomkoľvek systéme Windows bez potreby nainštalovaného Pythonu, čo zvyšuje kompatibilitu a dostupnosť aplikácie.

Kompilácia do spustiteľného súboru bola vykonaná pomocou nástroja PyInstaller vo verzii 6.6.0. Tento nástroj nie je súčasťou inštalácie Pythonu a je potrebné ho doinštalovať. Nástroj umožňuje jednoducho vytvárať spustiteľné súbory z Python skriptov pomocou jednoduchého príkazu v príkazovom riadku. Podporuje Windows, macOS a Linux, čo umožňuje vytvárať spustiteľné súbory pre rôzne operačné systémy. Nástroj analyzuje Python skript a automaticky zahrnie všetky potrebné knižnice a závislosti do výsledného spustiteľného súboru. To znamená, že výsledný .exe súbor obsahuje všetko potrebné na spustenie aplikácie.

Do elektronickej prílohy práce bol priložený vzorový PDF súbor (vzor.pdf), ktorý má poškodené kódovanie fontov v ňom využívaných. Konkrétne sa jedná o stranu 6 časopisu Praktická elektronika, vydanie 7 z roku 2018. Celé vydanie časopisu nemohlo byť priložené, nakoľko všetky vydania analyzovaných časopisov podliehajú autorským právam. Výsledný opravený PDF súbor je taktiež súčasťou elektronickej prílohy a jedná sa o súbor vzor_repaired.pdf. Pomocou týchto súborov je demonštrovaná funkčnosť opravného skriptu. Heš tohto vzoru nebol pridaný do JSON štruktúry, ktorú využíva overovací skript a preto by v prípade spustenia nad týmto súborom nedošlo k oprave. Vzorový súbor je potrebné opravovať iba pomocou spustiteľného súboru Type1toUnicode.exe. Pre opravu pomocou Type1toUnicode.exe je

možné použiť v príkazovom riadku príkaz:

Type1toUnicode.exe -p vzor.pdf -f to_unicode.json

Pred spustením je potrebné prejsť v príkazovom riadku do daného adresáru, kde sa spustiteľný súbor nachádza. V inom prípade je potrebné uviesť abstolútnu cestu k spustiteľnému súboru. Pokiaľ by bol heš súboru pridaný do JSON súboru magazine_hash.json, bolo by možné použiť spustiteľný súbor opravAR.exe nasledujúcim príkazom:

opravAR.exe -d ./ -j magazine_hash.json

V rámci diplomovej práce bol vytvorený zjednodušený návod pre používateľov, ako správne pracovať s vytvorenými skriptami:

1. Na lokálne úložisko skopírujte z originálnych CD/DVD všetky súbory, ktoré chcete opraviť. Najlepšie je zachovať pôvodnú adresárovú štruktúru pre lepšiu prehľadnosť po jednotlivých ročníkoch. Skript automaticky hľadá a opravuje PDF súbory vo všetkých podadresároch.
2. Do rovnakého adresáru uložte opravné skripty s príslušnými JSON štruktúrami. Konkrétne sa jedná o súbory:
   - **opravAR.exe** - spustiteľný súbor overovacieho skriptu.
   - **magazine_hash.json** - JSON štruktúra s hešmi elektronických časopisov.
   - **Type1toUnicode.exe** - spustiteľný súbor opravného skriptu.
   - **to_unicode.json** - JSON štruktúra s opravnými mapovaniami pre jednotlivé fonty.
3. Pomocou príkazového riadku spustite opravAR.exe s požadovanými argumentami. Pokiaľ nie sú parametre zadané správne, zobrazí sa do príkazového riadku pomocný popis, ktorý vysvetľuje použitie jednotlivých argumentov.
4. V danom umiestnení sa vytvoria opravené súbory s koncovkou „_repaired“ a adresár Log, ktorý obsahuje logovacie záznamy pre PDF súbory, ktoré skript opravoval. Pre každý súbor je vytvorený vlastný súbor, ktorého názov pozostáva z pôvodného názvu opraveného súboru a koncovky „_log“. Jedná sa o textové súbory s koncovkou .txt.

Opravný skript je možný spustiť aj samostatne bez overovacieho skriptu. Takýmto spustením však nie je možné zaručiť správnosť funkčnosti opravného kódovania, nakoľko sa nemusí jednať o testované vzorky. Skript je teda takto možné spustiť pre akékoľvek PDF dokumenty. Pokiaľ je funkcia skriptu správna, je možné kedykoľvek pridať heš súboru do JSON štruktúry, čo následne zabezpečí funkčnosť v prípade použitia overovacieho skriptu.

Pri PDF súboroch iných autorov nemusí kódovanie fontov kolidovať s opravným kódovaním v priloženom JSON súbore. Ide najmä o kódovanie znakov s diakritikou alebo špeciálnych znakov. Kódovanie znakov anglickej abecedy, čísel a interpunkč-

ných znamienok je však s vysokou pravdepodobnosťou rovnaké. Toto kódovanie súhlasilo aj pri iných PDF súboroch, ktoré využívali PostScriptové fonty typu 1.

Zverejnenie skriptov

Zdrojové kódy skriptov boli zverejnené na platforme GitHub v repozitári pod názvom Type1toUnicode. Zdrojový kód je teda voľne šírený ako otvorený nástroj. Vo vytvorenom repozitári sa vždy nachádza posledná verzia vytvorených skriptov a nemusí teda plne súhlasiť s verziami, ktoré boli popisované v rámci diplomovej práce. Repozitár aktuálne obsahuje nasledujúce súbory:

- **README.md** - vysvetľuje funkcionalitu skriptu a obsahuje návod na použitie pre užívateľa.
- **Type1toUnicode.py** - zdrojový kód aktuálnej verzie opravného skriptu.
- **Type1toUnicode.exe** - spustiteľný súbor kompilovaný z aktuálneho skriptu.
- **to_unicode.json** - JSON štruktúra s opravným kódovaním.
- **opravAR.py** - zdrojový kód aktuálnej verzie overovacieho skriptu.
- **opravAR.exe** - spustiteľný súbor kompilovaný z aktuálnej verzie overovacieho skriptu.
- **magazine_hash.json** - JSON štruktúra s hešmi súborov, ktoré skript zaručene opravuje.

Repozitár je verejne prístupný a jeho obsah môže vidieť ktokoľvek pomocou odkazu na repozitár[1].

[1] https://www.github.com/xgmitt00-220814/Type1toUnicode

Záver

V rámci diplomovej práce bolo vykonané dôkladné skúmanie vnútornej štruktúry PDF dokumentov s konkrétnym zameraním na problémy s kódovaním fontov a chýbajúce ToUnicode mapovania. Analyzované vzorové súbory, pochádzajúce z rôznych ročníkov časopisov Amatérske rádio a jeho podtitulov: Praktická elektronika, Konstrukční elektronika a Stavebnice a konstrukce, ukázali chyby v kódovaní znakov, ktoré boli zjavné najmä pri kopírovaní textu do iných aplikácií alebo vyhľadávania v texte v dôsledku chýbajúcich objektov ToUnicode pre správne kódovanie do Unicode.

Riešením týchto problémov bol vývoj skriptov v jazyku Python, zvoleného pre jeho jednoduchosť a širokú kompatibilitu. Skripty boli navrhnuté na poloautomatickú opravu kódovania všetkých fontov typu Type1 vo vzorových súboroch. Skript pre poloautomatickú opravu sa zameriava iba na opravu analyzovaných vzoriek. V rámci diplomovej práce bolo opravených 17 rôznych rodín fontov, čo pokrýva väčšinu fontov použitých v týchto dokumentoch. Funkčnosť skriptu bola overená na 196 vzorkách, čo zodpovedá dvanástim rokom vydaní časopisov Praktická elektronika a Konstrukční elektronika. Kľúčovým prvkom skriptu je generovanie objektu ToUnicode, ktorý správne mapuje glyfy na Unicode kódy. Táto funkčnosť bola zabezpečená pomocou záznamov v JSON súbore, ktorý obsahuje dvojice GID a príslušný Unicode kód.

Pri implementácii skriptu bolo využitých viacero knižníc, vrátane PyPDF, ktorá umožňuje čítanie, úpravu a manipuláciu s PDF dokumentmi, a knižníc logging a argparse na zaznamenávanie dôležitých informácií a spracovanie argumentov z príkazového riadku. Pre výpočty parametrov ukazujúcich podobnosť textových reťazcov boli využité existujúce knižnice pre Jaro-Winklerovu a Levenshteinovu vzdialenosť. Tieto nástroje umožnili efektívne riešenie problémov s kódovaním, ktoré sa vyskytli v analyzovaných dokumentoch.

Celkovým výsledkom diplomovej práce sú teda funkčné skripty, ktoré dokážu identifikovať a opraviť chyby v kódovaní fontov v PDF dokumentoch. Tieto skripty predstavujú prínos pre oblasť spracovania a opravy PDF dokumentov, umožňujúc vyhľadávanie a kopírovanie v dokumentoch.

Literatúra

[1] *PDF Reference sixth edition*[online]. [cit. 2023-12-01]. Dostupné z: <https://ghostscript.com/~robin/pdf_reference17.pdf>

[2] LUKAN, D. *PDF file format: Basic structure*[online]. [cit. 2023-12-01]. Dostupné z: <https://resources.infosecinstitute.com/topics/hacking/pdf-file-format-basic-structure/>

[3] SENAD, D. *PDF file format: Internal Document Structure Explained*[online]. [cit. 2023-12-01]. Dostupné z: <https://www.save-emails-to-pdf.com/news/pdf-file-format-internal-document-structure-explained/>

[4] FENNIAK, M. *The PDF Format*[online]. [cit. 2023-12-01]. Dostupné z: <https://pypdf2.readthedocs.io/en/3.0.0/dev/pdf-format.html>

[5] AL-SHARIF, Z.; ODEH, D.; AL-SALEH, M. *Towards Carving PDF Files in the Main Memory*[online]. [cit. 2023-12-01]. Dostupné z: <https://www.researchgate.net/figure/An-Example-of-a-PDF-Trailer_fig5_277622277>

[6] FENNIAK, M. *The PageObject Class*[online]. [cit. 2023-12-02]. Dostupné z: <https://pypdf.readthedocs.io/en/stable/modules/PageObject.html>

[7] *Chapter 1. PDF Syntax*[online]. [cit. 2023-12-02]. Dostupné z: <https://www.oreilly.com/library/view/developing-with-pdf/9781449327903/ch01.html>

[8] KREMER, K.H. *PDF Content Streams*[online]. [cit. 2023-12-02]. Dostupné z: <https://khkonsulting.com/2008/07/pdf-content-streams/>

[9] *Document*[online]. [cit. 2023-12-02]. Dostupné z: <https://pymupdf.readthedocs.io/en/latest/document.html>

[10] HANNA, K.T. *font*[online]. [cit. 2023-12-03]. Dostupné z: <https://www.techtarget.com/whatis/definition/font>

[11] BLISS, H. *What is a Bitmap Font?*[online]. [cit. 2023-12-03]. Dostupné z: <https://www.easytechjunkie.com/what-is-a-bitmap-font.htm>

[12] *Understanding outline fonts*[online]. [cit. 2023-12-03]. Dostupné z: <https://www.ibm.com/docs/en/zos/2.2.0?topic=fonts-understanding-outline>

[13] BARNHART, B. *The birth of Bézier curves and how it shaped graphic design*[online]. [cit. 2023-12-03]. Dostupné z: <https://www.linearity.io/blog/bezier-curves/>

[14] *Type 1 Font Format Supplement*[online]. [cit. 2023-12-03]. Dostupné z: <https://adobe-type-tools.github.io/font-tech-notes/pdfs/5015.Type1_Supp.pdf>

[15] *The history of fonts*[online]. [cit. 2023-12-03]. Dostupné z: <https://www.prepressure.com/fonts/basics/history#:~:text=When%20Adobe%20launched%20PostScript%20in,compression%20algorithm%20of%20font%20data>

[16] *Adobe Standard Encoding to Unicode*[online]. [cit. 2023-12-03]. Dostupné z: <http://ftp.unicode.org/Public/MAPPINGS/VENDORS/ADOBE/stdenc.txt>

[17] *TrueType overview*[online]. [cit. 2023-12-03]. Dostupné z: <https://learn.microsoft.com/en-us/typography/truetype/>

[18] SANTHANAM, L. *Font and Encoding Standard types supported in PDF for the representation of text content*[online]. [cit. 2023-12-03]. Dostupné z: <https://www.gnostice.com/nl_article.asp?id=383&t=Font_and_Encoding_Standard_types_supported_in_PDF_for_the_representation_of_text_content>

[19] HUDSON, J. *Introducing OpenType Variable Fonts*[online]. [cit. 2023-12-03]. Dostupné z: <https://medium.com/variable-fonts/https-medium-com-tiro-introducing-opentype-variable-fonts-12ba6cd2369>

[20] BERIO, D.; LEYMARIE, F.F.; ASENTE, P.; ECHEVARRIA, J. *StrokeStyles: Stroke-Based Segmentation and Stylization of Fonts*[online]. [cit. 2023-12-04]. Dostupné z: <https://research.gold.ac.uk/id/eprint/31944/1/strokestyles-opt.pdf>

[21] *Font flags*[online]. [cit. 2023-12-04]. Dostupné z: <https://www.verypdf.com/document/pdf-format-reference/pg_0458.htm>

[22] WHISTLER, K.; FREYTAG, A. *UNICODE CHARACTER ENCODING MODEL*[online]. [cit. 2023-12-11]. Dostupné z: <https://unicode.org/reports/tr17/>

[23] LOO, A. *Hash Function*[online]. [cit. 2024-04-15]. Dostupné z: <https://corporatefinanceinstitute.com/resources/cryptocurrency/hash-function/>

[24] DOBRAUNIG, C.; EICHLSEDER, M.; MENDEL, F. *Analysis of SHA-512/224 and SHA-512/256*[online]. [cit. 2024-04-18]. Dostupné z: <https://eprint.iacr.org/2016/374.pdf>

[25] *Jaro similarity*[online]. [cit. 2024-05-13]. Dostupné z: <https://rosettacode.org/wiki/Jaro_similarity>

[26] *What is Jaro-Winkler Similarity?*[online]. [cit. 2024-05-13]. Dostupné z: <https://www.baseclass.io/newsletter/jaro-winkler>

[27] NAM, E. *Understanding the Levenshtein Distance Equation for Beginners*[online]. [cit. 2024-05-14]. Dostupné z: <https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0>

Zoznam symbolov a skratiek

ACK Acknowledge

BDF Bitmap Distribution Format

BEL Bell

BS Backspace

CAN Cancel

CID Character Identification

CMYK Cyan, Magenta, Yellow, and Key

CR Carriage Return

DC Device Control

DEL Delete

DLE Data Link Escape

EM End of Medium

ENQ Enquiry

EOT End of Transmission

ESC Escape

ETB End of Transmission Block

ETX End of Text

FF Form Feed

FS File Separator

GID Glyph Identification

GS Group Separator

HT Horizontal Tabulation

IoT Internet of Things

ISO International Organization for Standardization

LF	Line Feed
NAK	Negative Acknowledge
NUL	Null
PCF	Portable Compiled Format
PDF	Portable Document Format
RGB	Red Green Blue
RS	Record Separator
SI	Shift In
SNF	Server Normal Format
SO	Shift Out
SOH	Start of Heading
SP	Space
STX	Start of Text
SUB	Substitute
SYN	Synchronous Idle
US	Unit Separator
VT	Vertical Tabulation

A Obsah elektronickej prílohy

```
/ ..................................... koreňový adresár priloženého archívu
├── Type1toUnicode.exe ..... spustiteľný súbor opravného skriptu
├── Type1toUnicode.py ..... zdrojový kód opravného skriptu
├── to_unicode.json ..... JSON štruktúra s opravným mapovaním
├── opravAR.exe ..... spustiteľný súbor overovacieho skriptu
├── opravAR.py ..... zdrojový kód overovacieho skriptu
├── magazine_hash.json ..... JSON štruktúra s hešmi časopisov
├── vzor.pdf ..... vzorový PDF súbor
└── vzor_repaired.pdf ..... opravený vzorový PDF súbor
```