

Univerzální bezztrátový archivátor

Universal Lossless Archiver

Bc. Ondřej Grim

Diplomová práce
2011



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

akademický rok: 2010/2011

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Ondřej GRIM**

Osobní číslo: **A09703**

Studijní program: **N 3902 Inženýrská informatika**

Studijní obor: **Informační technologie**

Téma práce: **Univerzální bezztrátový archivátor**

Zásady pro vypracování:

1. Vypracujte literární rešerši na téma univerzální bezztrátové archivátory.
2. Analyzujte požadavky na archivátory tohoto druhu.
3. Seznamte se s dostupnými nástroji pro tvorbu aplikací tohoto typu a vyberte nejvhodnější z nich.
4. Navrhněte vnitřní strukturu aplikace a uživatelské rozhraní.
5. Implementujte do svého programu vhodné kompresní algoritmy a rozhodovací mechanismy pro rozhodování použití konkrétního komprimačního postupu na libovolný typ dat.
6. Otestujte a porovnejte různé archivátory dle výsledného kompresního poměru a doby komprese a dekomprese.
7. Navrhněte možnosti dalšího vývoje a zdokonalení.

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. Morkes, D. Komprimační a archivační programy. Brno: Computer Press, 1998. 177 s. ISBN 80-7226-089-8
2. Salomon, D. Data Compression: The Complete Reference. 4 Ed. London: Springer-Verlag London Limited, 2007. 1092 s. ISBN 18-4628-602-6
3. Salomon, D. – Giovanni M. Handbook of Data Compression. 5 Ed. London: Springer-Verlag London Limited, 2010. 1359 s. ISBN 978-1-84882-902-2
4. Nelson M. – Gailly, J. The Data Compression Book. 2 Ed. New York: M&T Books, 1995. 541 s. ISBN 1-55851-434-1
5. Salomon, D. A Concise Introduction to Data Compression. London: Springer-Verlag London Limited, 2010. 310 s. ISBN 978-1-84800-071-1
6. Sayood, K. Introduction to Data Compression. 2 Ed. USA: Academic Press, 2000. 636 s. ISBN 1-55860-558-4
7. Blanchette, J. – Summerfield, M. C++ GUI Programming with Qt 4. 2 Ed. Westford, Massachusetts: Trolltech ASA, 2008. 752 s. ISBN-13: 978-0-13-235416-5
8. Summerfield, M. Advanced Qt Programming: Creating Great Software with C++ and Qt 4. Westford, Massachusetts: Addison-Wesley, 2010. 536 s. ISBN 978-0-321-63590-7

Vedoucí diplomové práce:

Ing. Tomáš Sysala, Ph.D.

Ústav automatizace a řídicí techniky

Datum zadání diplomové práce:

24. února 2011

Termín odevzdání diplomové práce:

18. května 2011

Ve Zlíně dne 24. února 2011

prof. Ing. Vladimír Vašek, CSc.

děkan

L.S.

doc. Mgr. Roman Jašek, Ph.D.

ředitel ústavu

ABSTRAKT

Tato diplomová práce se zabývá návrhem a implementací univerzálního bezztrátového archivátoru. To znamená aplikace sloužící pro kompresi dat, která jsou jí předána. Také popisuje implementaci rozhodovacího mechanismu, který v archivátoru zajišťuje volbu nejvhodnějšího kompresního algoritmu. Práce čtenáře seznamuje s problematikou spojenou s tvorbou archivačních programů a s možným způsobem implementace takového typu programu.

KLÍČOVÁ SLOVA

C++, Qt, archivátor, bezztrátová komprese, dekomprese, extrakce, kompresní algoritmy, RLE, LZW, Huffmanovo kódování, grafické uživatelské rozhraní, GUI

ABSTRACT

This thesis describes the design and implementation of universal lossless archiver. It means the application, which is used to compress data that are passed into this application. It also describes the implementation of the decision-making mechanism in the archiver that provides the choice of the most appropriate compression algorithm. The readers are introduced with the problems, which are associated with creating repositories of programs and with potential way of implementation of this type of program.

KEYWORDS

C++, Qt, archiver, lossless compression, decompression, extraction, compression algorithms, RLE, LZW, Huffman coding, graphical user interface, GUI

Rád bych poděkoval svému vedoucímu Ing. Tomáši Sysalovi, Ph.D., bez něhož by tato diplomová práce nikdy nevznikla. Současně bych chtěl tímto způsobem poděkovat všem, kteří mě po dobu, kdy tato diplomová práce vznikala, jakýmkoliv způsobem podporovali.

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD.....	10
I TEORETICKÁ ČÁST	11
1 POJEM „ARCHIVÁTOR“	12
2 KOMPRESSE	13
2.1 BEZZTRÁTOVÉ A ZTRÁTOVÉ KOMPRESNÍ ALGORITMY	13
2.1.1 Bezztrátové kompresní algoritmy	13
2.1.2 Ztrátové kompresní algoritmy	13
2.2 DALŠÍ ROZDĚLENÍ KOMPRESNÍCH ALGORITMŮ	16
3 DATOVÉ SOUBORY	18
3.1 ZÁKLADNÍ RYSY	18
3.2 PROBLEMATIKA BLOKŮ PŘI UKLÁDÁNÍ DAT NA DISKU	18
3.3 TYPY SOUBORŮ	19
4 UŽIVATELSKÁ ROZHRAŇÍ.....	21
4.1 DRUHY KOMUNIKACE ČLOVĚKA S POČÍTAČEM	21
4.1.1 Dávkové zpracování	21
4.1.2 Příkazový řádek (CLI)	21
4.1.3 Textové uživatelské rozhraní (TUI)	23
4.1.4 Grafické uživatelské rozhraní (GUI)	23
4.1.5 Multimediální rozhraní.....	24
II PRAKTICKÁ ČÁST	26
5 PRAKTICKÝ ÚVOD	27
5.1 ZAVEDENÍ POJMŮ	27
5.2 POŽADAVKY NA APLIKACI	28
5.3 POUŽITÉ TECHNOLOGIE	29
6 NÁVRH	31
6.1 GRAFICKÉ UŽIVATELSKÉ ROZHRAŇÍ	31
6.1.1 Hlavní okno aplikace.....	31
6.1.2 Nastavení aplikace.....	34
6.1.3 Nastavení parametrů archivace	35
6.1.4 Nastavení poznámky k archivu	38
6.1.5 Načítání zdrojů	39
6.1.6 Archivační/extrakční dialogové okno	39
6.1.7 Zadáání hesla k archivu.....	41
6.1.8 O aplikaci	41
6.2 DATOVÉ STRUKTURY.....	42
6.2.1 Struktura „Folder“:	42
6.2.2 Struktura „File“:.....	43
6.2.3 Struktura „MethodChoose“:	44

6.3	ARCHIV	45
6.3.1	Obecné vlastnosti	45
6.3.2	Obsah archivu.....	45
6.3.3	Optimalizace dat v archivu.....	47
7	OPERACE S ARCHIVEM.....	49
7.1	POLOŽKY V APLIKACI	49
7.2	NAČTENÍ POLOŽEK	49
7.2.1	Načtení souborů.....	49
7.2.2	Načtení složky	50
7.3	PROCES ARCHIVACE	51
7.3.1	Hlavní cyklus.....	52
7.3.2	Záznam struktury „Folder“	53
7.3.3	Záznam struktury „File“.....	54
7.4	NAČTENÍ ARCHIVU	54
7.5	PROCES EXTRAKCE.....	56
7.5.1	Extrakce kořenové složky (struktura „Folder“)	56
7.5.2	Extrakce kořenového souboru (struktura „Folder“).....	56
7.5.3	Extrakce vnořené složky (struktura „File“).....	56
7.5.4	Extrakce vnořeného souboru (struktura „File“).....	57
7.6	DOČASNÉ SOUBORY	57
8	OSTATNÍ IMPLEMENTACE	59
8.1	IMPLEMENTOVANÉ METODY	59
8.1.1	Bez komprese	59
8.1.2	Run-Length Encoding (RLE)	59
8.1.3	Adaptivní Huffmanovo kódování.....	60
8.1.4	Lempel-Ziv-Welch (LZW).....	60
8.2	ROZHODOVACÍ MECHANISMUS	61
8.2.1	Vzorek souboru	61
8.2.2	Nastavení velikosti vzorku	62
8.2.3	Princip mechanismu	62
8.2.4	Optimalizace	63
8.3	STROMOVÁ STRUKTURA	64
8.3.1	Shrnutí pojmů.....	64
8.3.2	Uchování	65
8.3.3	Rekonstrukce.....	66
8.3.4	Zobrazení položek v náhledu aplikace	69
8.4	VLÁKNA	70
8.5	ROZLIŠENÍ SLOŽEK A SOUBORŮ V POHLEDU.....	71
8.6	ZABEZPEČENÍ ARCHIVU	72
8.7	BLOKACE NEAKTUÁLNÍCH FUNKCÍ.....	72
9	ZHODNOCENÍ	74

9.1	SHRNUTÍ VÝSLEDKŮ IMPLEMENTACE.....	74
9.2	SROVNÁNÍ APLIKACE S VYBRANÝMI ARCHIVAČNÍMI PROGRAMY.....	74
9.2.1	Základní informace	75
9.2.2	Test 1 – spustitelný soubor.....	76
9.2.3	Test 2 – 24-bitový BMP soubor	77
9.2.4	Test 3 – dokument DOC	78
9.2.5	Test 4 – textový soubor	79
9.2.6	Test 5 – 20 malých textových souborů.....	80
9.2.7	Test 6 – přídatné informace:.....	81
9.3	ZHODNOCENÍ DOSAŽENÝCH VÝSLEDKŮ.....	81
9.4	ZHODNOCENÍ DALŠÍHO VÝVOJE APLIKACE.....	82
ZÁVĚR		84
CONCLUSION		85
SEZNAM POUŽITÉ LITERATURY		86
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK		87
SEZNAM OBRÁZKŮ		88
SEZNAM TABULEK.....		89
SEZNAM PŘÍLOH.....		90

ÚVOD

Každou vteřinu vznikají neustále další informace, které je nutné elektronicky uchovávat. Tyto informace jsou uchovávány v podobě dat (souborů) na různých záznamových médiích. Z pohledu procesu zálohování není nijak podstatná důležitost dat. Může se totiž jednat o simulaci reakce světového trhu na ekonomickou krizi, údaje o globálním oteplování nebo může jít třeba jen o fotografii z firemního večírku nebo o školní projekt. Také není nijak podstatné, zda-li se jedná o data unikátní (vytvořený hudební soubor, sestříhané video, prezentace k novému projektu atd.) nebo duplicitní (instalátory aplikací, studijní materiály atd.). Naopak je při procesu zálohování velice důležité, kolik tato data zabírají kapacitního prostoru. S velikostí dat totiž rostou požadavky na datová média, čímž se zvyšují finanční náklady vynaložené na jejich zálohu. Při zachování stejné propustnosti datové linky roste s velikostí dat i časová náročnost datového přenosu. Z těchto důvodů došlo k rozvoji tzv. kompresních algoritmů, které dokážou snížit velikost souborů. Kompresní algoritmy se primárně dělí do dvou skupin na základě principu, jakým dosahují požadovaného výsledku. První skupinu tvoří algoritmy sloužící pro ztrátovou kompresi, při které dochází k „ořezání“ souborů o nadbytečné informace. Do druhé skupiny patří algoritmy pro bezztrátovou kompresi, kde ze zkomprimovaného souboru lze zpětně vytvořit identický původní soubor. Žádný z kompresních algoritmů není univerzální a každý je vhodný pro jiný typ dat. Tato diplomová práce se bude zabývat tvorbou univerzálního bezztrátového archivátoru – aplikace, která bude mít implementováno několik bezztrátových komprimačních algoritmů, mezi kterými bude volit ten nejvhodnější algoritmus pro daný typ vstupních dat.

V první části této práce se zaměřím na teoretické předpoklady pro návrh aplikace tohoto typu.

Ve druhé části se detailněji zaměřím na popis samotné implementace univerzálního bezztrátového archivátoru a použitých kompresních metod.

I. TEORETICKÁ ČÁST

1 POJEM „ARCHIVÁTOR“

Na úplném začátku je vhodné definovat, co to vlastně znamená pojem „archivátor“. Jedná se o počítačový program vytvořený v libovolném programovacím jazyce, který využívá zpravidla bezztrátové kompresní algoritmy za účelem snížení velikosti jednoho nebo více souborů, které jsou mu předány. Výstupem archivačního procesu je jeden soubor, tzv. archiv. Ten v sobě zapouzdřuje všechny zkomprimované soubory a složky, a to včetně jejich hierarchické struktury. Některé archivátory umožňují výstupní archiv rozdělit na menší části, což lze s výhodou použít při snaze o nakopírování velkého archivu na média s nižší fyzickou datovou kapacitou (např. CD, DVD, paměťové karty nebo USB flashdisky). Pro práci s archivem musí být používán stejný archivační program, ve kterém byl archiv vytvořen. Je však možné použít i takový archivační program, který využívá stejné kompresní/dekompresní algoritmy a vytváří archiv stejným způsobem, jako původní archivátor. Jakákoliv informace (byť jediný bajt), která je při archivačním procesu přidána navíc, může znamenat, že v jiném archivačním programu může být archiv načten nesprávně. Soubory se v archivu nacházejí ve zkomprimované podobě a před jejich spuštěním musí být z archivu extrahovány (rozbaleny), čímž dojde k jejich dekompresi.

2 KOMPRESSE

Kompresi (nebo také komprimaci) dat lze obecně definovat jako operaci nad souborem, jejímž cílem je zmenšení datového objemu tohoto souboru. Toho se s výhodou využívá zejména při archivaci dat, kdy lze kompresí snížit ekonomické náklady (např. cena médií a náklady na úložní prostory médií) s archivací spojené a zlepšit přístupnost a přehlednost archivovaných dat (menší počet záložních médií). Další oblastí, kde se komprese hojně využívá, jsou datové přenosy. Zde snížení velikosti přenášených dat, a tím pádem i snížení doby potřebné pro jejich přenos, bývá klíčové zejména u sítí s omezenou šířkou přenosového kanálu. Jednoznačným příkladem je mobilní telefonie. V neposlední řadě lze kompresi využít také pro úsporu místa na datovém médiu tím, že je větší množství malých souborů vloženo do jednoho velkého archivu (více informací ohledně této problematiky viz kapitola 3.2). Kompresi lze rozdělit dle různých kritérií:

2.1 Bezztrátové a ztrátové kompresní algoritmy

V závislosti na ztrátovosti vznikající v průběhu kompresního procesu se algoritmy dělí na tzv. ztrátové a bezztrátové:

2.1.1 Bezztrátové kompresní algoritmy

Zpravidla nedosahují takové míry komprese jako algoritmy ztrátové. Bezztrátové algoritmy vychází z předpokladu, že je nepřípustná sebemenší ztráta dat. Komprimovaný soubor musí být před spuštěním dekomprimován, tzn., že ze zkomprimovaného obsahu souboru musí být zrekonstruována jeho původní podoba. „Žádný komprimační program nedokáže bezztrátovou metodou komprimace komprimovat všechny soubory o velikosti větší nebo rovné N bitů pro libovolné celočíselné N větší nebo rovno 0.“[1] To znamená, že „pro jakýkoliv komprimační algoritmus lze vymyslet sadu vstupních souborů, které tento algoritmus nebude schopen zmenšit ani o jediný bit, aniž by došlo ke ztrátě původních informací.“[1] Z výše uvedeného vyplývá, že žádný bezztrátový kompresní algoritmus není univerzální pro všechny typy vstupních dat.

2.1.2 Ztrátové kompresní algoritmy

Vychází z předpokladu, že některé informace jsou v souboru nadbytečné a může tedy dojít k jejich odfiltrování, aniž by daný soubor ztratil svoji vypovídající hodnotu. Ztrátová

komprese se využívá u obrazových a zvukových záznamů. Míra komprese je závislá na velikosti filtrovaného pásma, tzn. oblasti, která bude kompresním algoritmem odstraněna. S rostoucí mírou komprese sice dochází ke zmenšování velikosti souboru, ale současně dochází i k jeho „deformaci“ – snížení kvality. Je-li komprese prováděna s ohledem na fyziologické vlastnosti lidského sluchu a/nebo zraku (v závislosti na konkrétním typu multimediálního souboru), je rozdíl mezi zdrojovou a ztrátově komprimovanou verzí multimediálního souboru lidskými smysly (téměř) nerozlišitelný. V opačném případě má výstupní soubor sice malou velikost, ale je silně zdeformován. Tím pádem se potom stává v některých případech, např. při orientaci na detail nebo při zobrazování na větších plochách, nepoužitelným. Výše popsané chování komprese lze jednoznačně prezentovat na příkladě srovnání obrázku plné kvality s jeho obrazem při provádění ztrátové komprese:



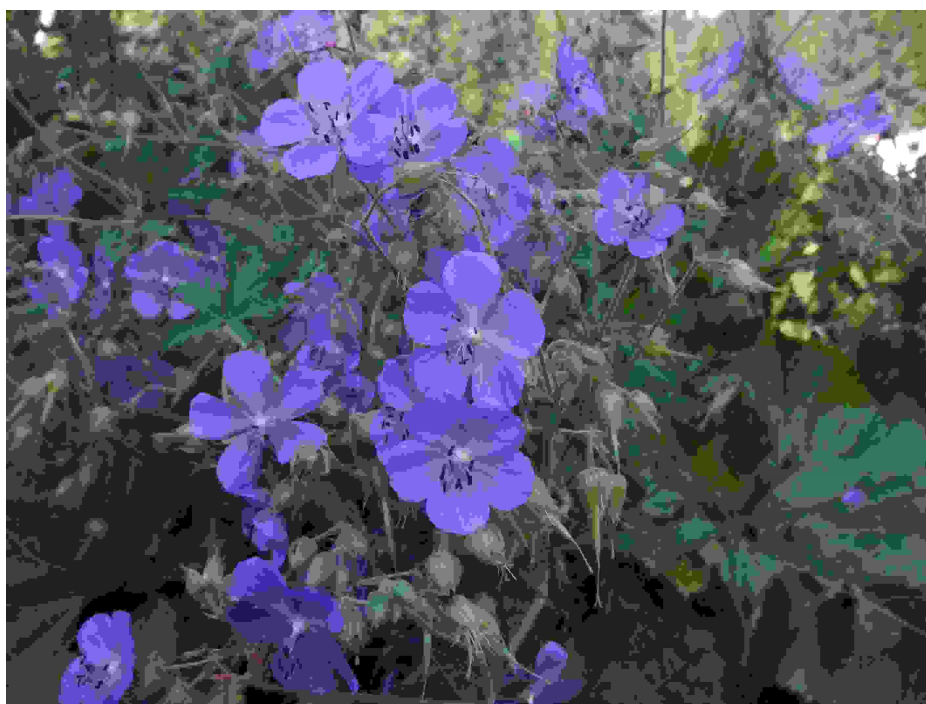
Obr. 1. Originální obrázek v plné kvalitě

Na Obr. 1. je vidět obrázek ve stoprocentní kvalitě. Jeho velikost je rovna 2 933 617 bajtů. Je-li tento obrázek podroben kompresi, při níž je jeho kvalita snížena na 50%, dochází k patrným vizuálním změnám, které jsou způsobeny postupnou segmentací obrazu na bloky (viz Obr. 2.). Tyto změny způsobené kompresí jsou viditelné zejména při zvětšení obrázku. Velikost komprimovaného souboru je 325 215 bajtů, tzn. 9x nižší než velikost originálu. Extrémně vysokou kompresí, při níž dojde ke snížení kvality na 1%, je pak

dosaženo snížení velikosti obrázku na 54 883 bajtů, tj. 53,45krát nižší velikost oproti originální verzi. Takto komprimovaný obrázek ovšem ztrácí, vzhledem k určitým typům užití (např. pro další botanické bádání nebo pro uchování vzpomínek z dovolené), svoji vypovídací hodnotu (viz Obr 3.).



Obr. 2. Komprimovaný soubor, kvalita 50%



Obr. 3. Komprimovaný soubor, kvalita 1%

Z výše uvedeného příkladu vyplývá, že vhodně zvolenou kompresí lze snížit velikost souboru a šetřit tak prostor na paměťovém médiu. Naopak příliš vysokou kompresí dojde ke znehodnocení obrazu. Míra komprese by se proto měla volit s přihlédnutím k nedokonalosti lidských smyslů, tzn. zraku a/nebo sluchu, a k následnému použití komprimovaného souboru.

Ztrátově zkomprimovaný soubor odfiltrované informace nenávratně ztrácí a již nelze žádným způsobem zrekonstruovat jeho původní podobu.

2.2 Další rozdělení kompresních algoritmů

Dále lze kompresní algoritmy rozdělit do skupin podle následujících kritérií:

- **Kompresse beroucí ohled na strukturu komprimovaných dat:**
 - **Fyzická** – komprese probíhá bez kladení důrazu na logickou strukturu dat.
 - **Logická** – při kompresi je brána v potaz logická struktura dat, přičemž jsou sekvence znaků nahrazovány jednotlivými znaky nebo kratšími sekvencemi znaků. Příkladem logické komprese je například zkratka ČEDOK (5 znaků), která v sobě skrývá dlouhý název „Československá dopravní kancelář“ (32 znaků).
- **Dle univerzálnosti:**
 - **Univerzální** – kompresní algoritmy jsou použitelné pro data libovolného typu.
 - **Speciální** – kompresní algoritmy speciálně vytvořené a uzpůsobené pro konkrétní typy souborů, např. pouze pro obrazové či zvukové záznamy atd. Zpravidla dosahují lepších kompresních výsledků (nižší datové objemy zkomprimovaných souborů a/nebo nižší časy potřebné k provedení komprese) než univerzální algoritmy.
- **Dle symetričnosti:**
 - **Symetrické** – časová náročnost komprese i dekomprese souboru je prakticky shodná.
 - **Asymetrické** – časová náročnost komprese a dekomprese je rozdílná. Skutečnost, jestli je dekompresní proces rychlejší nebo pomalejší než proces kompresní, záleží na konkrétním algoritmu. Ten se zpravidla volí v závislosti na tom, jak často se bude s daty v budoucnu pracovat.

- **Dle adaptivnosti:**
 - **Neadaptivní** – využívají se pro specifické typy dat. Obsahují totiž předdefinované a neměnné slovníky nebo řetězce znaků, u nichž se předpokládá, že jejich výskyt v komprimovaném souboru je nejvyšší.
 - **Adaptivní** – slovník je vytvořen až na základě obsahu vstupního souboru.
 - **Semiadaptivní** – jedná se dvouprůchodový kompresní proces. V první fázi dojde k průchodu celého souboru a k sestavení slovníku. Ve druhé fázi již dochází k samotné kompresi, která probíhá za pomoci slovníku, který byl vytvořen v předchozí fázi.
- **Dle vstupních dat:**
 - **Blokové** – na vstup algoritmu jsou přiváděna data jako celek (blok). Výstup je také blokový.
 - **Proudové** – na vstup je přiváděn obsah souboru po jednotlivých znacích. Stejným způsobem je také řešen výstup algoritmu.

3 DATOVÉ SOUBORY

Soubor lze, mimo jiné, definovat z pohledu informačních technologií jako logickou posloupnost bajtů prezentovanou jako jednotný a samostatný celek uložený na zvoleném datovém médiu v několika blocích. Tyto bloky mají určitou (a stejnou) velikost. Ta závisí na velikosti datového média a na použitém souborovém systému. Jednotlivé datové bloky jsou v rámci média uloženy dle předem stanovených pravidel a není nutností, aby byly řazeny bezprostředně za sebou. Pravidla, na základě kterých dochází k ukládání jednotlivých bloků souboru na disk, a způsob, jakým jsou tyto bloky v případě potřeby načítány, závisí na konkrétním operačním systému. Řadového uživatele ani zkušeného vývojáře tento mechanismus nemusí zajímat.

3.1 Základní rysy

Soubor je jednoznačně definován jménem, příponou (neplatí vždy, mohou se vyskytovat i soubory bez přípon), velikostí a případnými atributy operačního systému, které soubor blíže specifikují (např. skrytý, jen pro čtení, systémový,...). Na základě přípony souboru lze obvykle odhadnout, co je jeho obsahem. Například přípona „*exe*“ značí spustitelný (anglicky „*executable*“) soubor, „*bmp*“ označuje bitmapový obrázek atd. Tento fakt usnadňuje uživateli práci se souborovým systémem.

3.2 Problematika bloků při ukládání dat na disku

Z pohledu ukládání souborů v rámci daného souborového systému je velmi podstatnou skutečností, že soubor na datovém médiu nezabírá pouze kapacitu rovnu své velikosti, ale N krát velikost bloku, kde N je počet bloků, na které je soubor rozdělen. Vyplývá to z faktu, že je-li do jednoho bloku zapsána jakákoliv informace (byť jediný bajt), nelze do tohoto bloku zapisovat žádné další informace. To lze demonstrovat na jednoduchých příkladech:

Příklad 1:

Uvažujme, že prostor datového média je rozdělen na jednotlivé bloky o velikosti 4096 bajtů. Dále uvažujme soubor o velikosti 4100 bajtů. 4096 bajtů tohoto souboru je zapsáno do jednoho bloku. Zbývají ovšem 4 bajty, které musí být zapsány do bloku dalšího. Navzdory tomu, že v bloku dosud zbývá 4092 bajtů, nelze do něj zapisovat a celý

blok je přiřazen souboru, jehož část je v něm uložena. To znamená, že soubor o velikost 4100 bajtů bude na datovém médiu zabírat $2 * 4092 \text{ bajtů} = 8184 \text{ bajtů}$.

Příklad 2:

Uvažujme velikost bloku 4096 bajtů. Dále uvažujme 4000 malých souborů o velikosti 1 bajt (uvažujme, že v každém souboru je uložen 1 znak datového typu Char). Každý z těchto souborů zabírá na datovém médiu 4096 bajtů. $4000 \times 4096 = 16\,384\,000$ bajtů => přes 16 MB paměťového prostoru média. Kdyby byl obsah všech souborů shrnut do jediného souboru, jednalo by se o soubor o velikosti 4000 bajtů a zabírající 4096 bajtů. Oproti výše zmiňovaným 16-ti MB se jedná o markantní úsporu paměťového prostoru. Při archivaci je důležité brát v úvahu i přídatné informace, které kromě obsahů souborů nesou také informace o jejich názvech, velikostech, hierarchii atd. Ale i přes tuto skutečnost bude archiv vytvořený ze 4000 souborů o velikosti 1 bajt zabírat naprosto zanedbatelnou kapacitu datového média oproti původní 16-ti MB podobě těchto souborů.

Na závěr pojednání o datových blocích a jejich využití je ještě nutné zmínit, že některé souborové systémy podporují tzv. „*tail picking*“, což je metoda umožňující zapisovat do datových bloků, které již nějakou informaci obsahují.

3.3 Typy souborů

Pro účely této práce uvažujme jedno ze základních dělení souborů, a to rozdělení na soubory textové a binární.

Tvrzení inženýra Herouta:

„Textové soubory mají tu velkou výhodu, že je možné si jejich obsah kdykoliv prohlédnout, vytvořit nebo opravit běžným editorem. Jejich nevýhodou ale je, že pro uchování stejného množství informace potřebují mnohem více prostoru. Například číslo 65535 zabere v textovém souboru prostor 5-ti bajtů, zatímco v binárním souboru třeba jen 2 bajty.“[2]

Rozbor tvrzení:

Pro následující vysvětlení uvažujme 32 bitový systém, datové typy jazyka C/C++ a použití znakové sady UTF-8, kde je pro uložení každého znaku potřebný 1 bajt paměťového prostoru.

Textový soubor je vlastně souborem znaků datového typu Char (neboli znak). Každý takový znak zabírá kapacitní prostor o velikosti 1 bajt. Číslo 65535 se skládá z 5-ti znaků typu Char. To znamená, že textový soubor s tímto obsahem bude zabírat 5 kB kapacitního prostoru (uvažujeme-li skutečnou velikost souboru, nikoli velikost souboru na datovém médiu). Při práci s binárními soubory však lze do těchto souborů zapisovat také data jiných datových typů, např. logickou hodnotu Boolean, číselnou hodnotu Integer atd. Pro uložení číselné hodnoty 65535 plně postačí proměnná datového typu Unsigned short. Pro tento datový typ je v paměti alokován prostor 2 bajty (2^{16} bitů). Jelikož se jedná o neznaménkový datový typ, tzn., že nejvýznamnější bit¹ neslouží pro uchování informace o znaménku, ale pro uchování číselné hodnoty, lze prostřednictvím tohoto datového typu uchovat číselnou hodnotu v rozsahu 0 – 65535. Při zápisu hodnoty do binárního souboru pak není nutné ukládat ASCII reprezentace jednotlivých znaků, ale uloží se přímo binární hodnota celého čísla obsažená v datovém typu Unsigned short, tzn. 1 x 2 bajty. Srovnáním datových objemů nutných k uložení téže informace, avšak reprezentované jednou textovým a podruhé binárním souborem, lze dospět k poznatku, že binární soubor je při uchování stejné informace 2,5 krát menší než soubor textový.

Další výhodou binárních souborů je, že „se s nimi pracuje mnohem rychleji, než s textovými soubory. Důvody jsou dva – jednak jsou binární soubory kratší a jednak při zápisu čísla do textového souboru je nutné provést jeho konverzi z vnitřní reprezentace čísla v počítači na textovou podobu, což je časově náročné (u čtení dochází také ke konverzi, ale k obrácené – z textu na číslo). Tyto konverze v binárních souborech odpadají, protože se do nich zapisuje přímo obsah paměti po bajtech.“[2]

Z výše uvedených důvodů se binárním souborům dává přednost před soubory textovými v případech, kdy dochází ke snaze ušetřit datový prostor potřebný pro uložení dat, kdy je podstatná rychlost práce se souborem a kdy se neočekává přímý zásah² uživatele do tohoto souboru.

¹ Anglicky „*Most Significant Bit*“ neboli „*MSB*“.

² Např. prostřednictvím editoru textových souborů.

4 UŽIVATELSKÁ ROZHRAŇÍ

Typ uživatelského rozhraní určuje způsob komunikace mezi dvěma subjekty, kterými mohou být člověk (v prostředí obsluhy počítačů se pro něj vžil termín „uživatel“) a počítač. Tato komunikace má formu zadávání příkazů, klávesových zkratk, vyplňování formulářů atd. Typy možných uživatelských akcí jsou závislé na konkrétním typu uživatelského rozhraní. Uživatelské akce vyvolávají jim odpovídající naprogramované reakce počítače, kterými mohou být např. zobrazení výsledku nějaké uživatelské akce nebo např. varování vztažené k výskytu nějaké události, jako je třeba smazání souboru. Reakce počítače mohou mít i formu zvukového signálu. Jednotlivé reakce mohou být, v závislosti na implementaci aplikace, mezi sebou vzájemně kombinovány.

4.1 Druhy komunikace člověka s počítačem

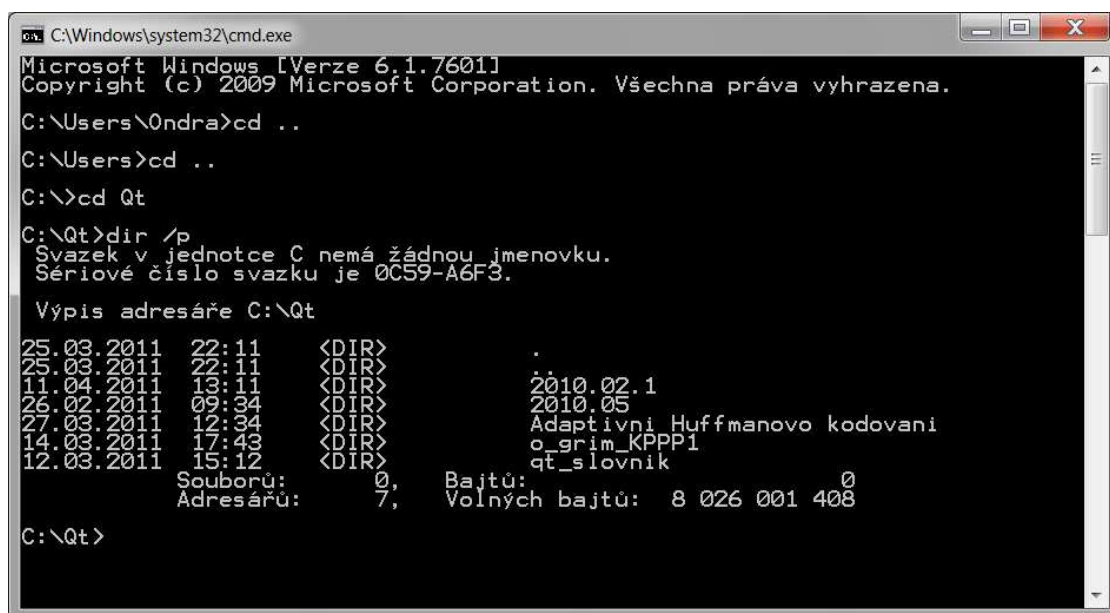
4.1.1 Dávkové zpracování

Anglicky „*Batch Processing*“. Dávkový způsob komunikace s počítačem vznikl přibližně kolem roku 1950. Před samotným zahájením komunikace je uživatelem vytvořena tzv. dávka – soubor po sobě jdoucích instrukcí. Po zahájení dávkového zpracování vytvořené dávky již zpravidla není možné posloupnost instrukcí nijak ovlivnit (neuvažujeme-li přerušení zpracovávání dávky nebo situaci, kdy je uživateli nabídnuta možnost volby).

4.1.2 Příkazový řádek (CLI)

Zkratka je odvozena z anglického „*Command Line Interface*“. V roce 1969 se objevil nový způsob komunikace s počítačem, tzv. příkazový řádek (nebo také příkazová řádka). Uživatel s počítačem, resp. s operačním systémem nebo programem, komunikuje prostřednictvím jednoduchých příkazů. Lze konstatovat, že rozdíl mezi příkazovým řádkem a dávkovým zpracováním je v tom, že komunikace s počítačem probíhá jistou formou „interaktivního dialogu“, tzn., že uživatel má možnost dynamicky reagovat na výsledek předchozí operace. Příkazy příkazového řádku mohou být doplněny o jeden

nebo více parametrů³, které blíže specifikují jeho funkci (např. způsob zobrazení výsledku). Prostřednictvím příkazového řádku lze zpracovávat v jeden okamžik pouze jeden příkaz (nebo sadu příkazů). Dokud není zpracování předchozího příkazu/sady příkazů dokončeno, není uživateli umožněno zadávat příkazy další. Připravenost systému k přijetí příkazů zadávaných uživatelem do příkazového řádku je indikována např. blikáním kurzoru. Tato skutečnost však není podmínkou. Způsob, jakým je uživatel informován o připravenosti systému, závisí na nastavení příslušných parametrů operačního systému, který příkazový řádek poskytuje. Operace odeslání příkazu je vyvolána stisknutím klávesy „ENTER“, přičemž dojde k jeho interpretaci. Je-li příkaz syntakticky v pořádku, je operačním systémem proveden. V opačném případě je uživatel upozorněn na nesprávný zápis (neznámý příkaz, neznámý parametr atd.). S příkazovým řádkem uživatel pracuje primárně prostřednictvím klávesnice. Podpora myši je závislá na konkrétní implementaci příkazového řádku. Je-li podpora myši implementována, zpravidla se využívá jen pro označování a vkládání textu. Práce s příkazovým řádkem po uživateli vyžaduje jistou znalost příkazů, jejich syntaxe, popř. i význam jednotlivých parametrů těchto příkazů. Vzhled příkazové řádky je prezentován na Obr. 4.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Verze 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Všechna práva vyhrazena.

C:\Users\Ondra>cd ..
C:\Users>cd ..
C:\>cd Qt
C:\Qt>dir /p
Svazek v jednotce C nemá žádnou imenovku.
Sériové číslo svazku je 0C59-A6F3.

Výpis adresáře C:\Qt

25.03.2011  22:11    <DIR>          .
25.03.2011  22:11    <DIR>          ..
11.04.2011  13:11    <DIR>          2010.02.1
26.02.2011  09:34    <DIR>          2010.05
27.03.2011  12:34    <DIR>          Adaptivní Huffmanovo kodování
14.03.2011  17:43    <DIR>          o_grim_KPPP1
12.03.2011  15:12    <DIR>          qt_slovník
                Souborů:          0,      Bajtů:          0
                Adresářů:        7,      Volných bajtů: 8 026 001 408

C:\Qt>
```

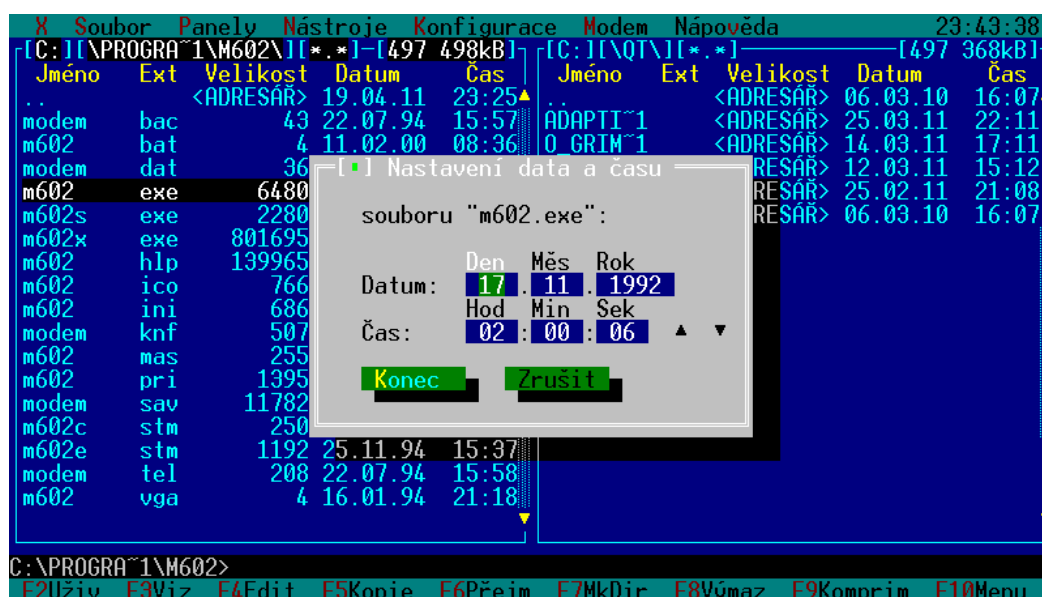
Obr. 4. Příklad vzhledu příkazové řádky

³ Závisí na implementaci konkrétního příkazu.

4.1.3 Textové uživatelské rozhraní (TUI)

Zkratka odvozena z anglického „Text User Interface“ (někdy také „Textural User Interface“). Jedná se o určitý přechod mezi příkazovým řádkem a grafickým uživatelským rozhraním. Při tvorbě textového uživatelského rozhraní se vychází z předpokladu, že použitá sada znaků obsahuje, mimo jiné, také speciální znaky, s jejichž pomocí lze na obrazovce jednoduše vytvořit různé tvary představující tabulky, tlačítka atd. Obrazovka je brána jako rastr o pevně daném počtu řádků a sloupců, přičemž každé políčko tohoto rastru může obsahovat pouze jeden znak (symbol) z použité sady znaků. Pro každé políčko lze také definovat barvu přiřazené hodnoty (znaku z tabulky znaků) a barvu pozadí.

Součástí textového uživatelského rozhraní zpravidla bývá příkazový řádek, Tyto dva způsoby komunikace s počítačem se vzájemně doplňují. Mimo klávesnici je zde již zpravidla implementována také podpora myši. Vzhled textového uživatelského rozhraní představuje Obr. 5.



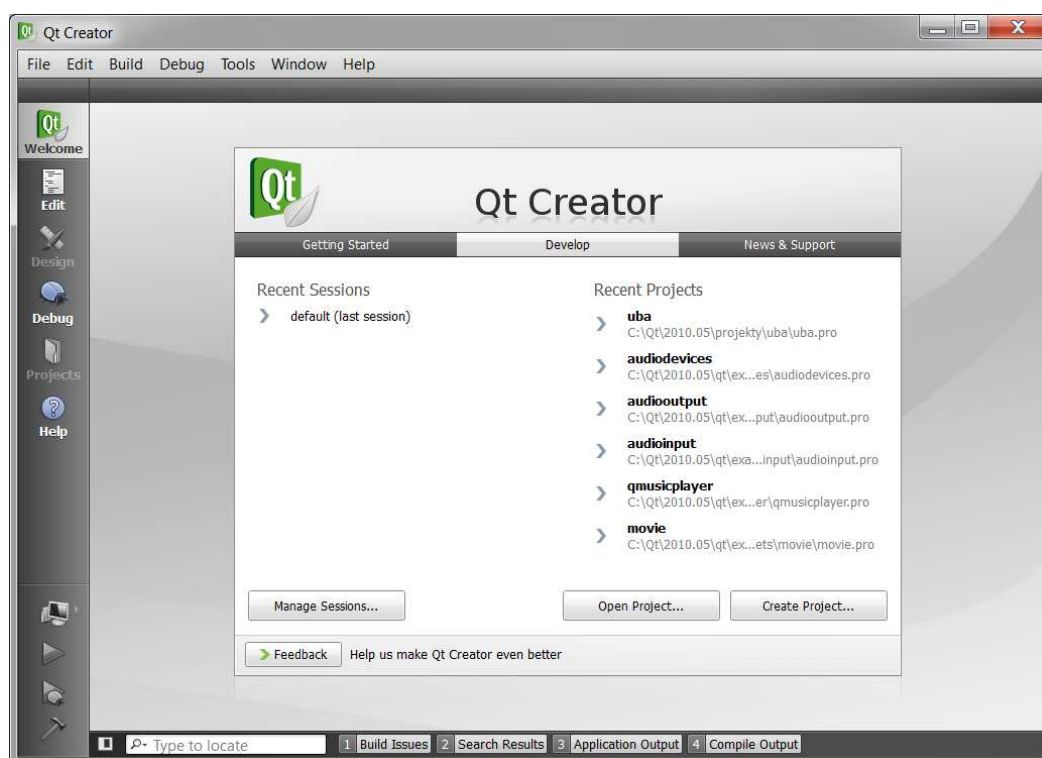
Obr. 5. Příklad vzhledu textového uživatelského rozhraní

4.1.4 Grafické uživatelské rozhraní (GUI)

Zkratka odvozena z anglického „Graphical User Interface“. S trochou nadsázky by se dalo říci, že grafické uživatelské rozhraní je jakousi „stavebnicí“. Základním stavebním prvkem je okno – plocha, na kterou se přidávají jednotlivé stavební kameny různých typů (se svými specifickými vlastnostmi a funkcemi), které jsou danému oknu podřízeny. K ovládání

grafického uživatelského rozhraní jsou využívána různá polohovací zařízení, zejména se pak GUI ovládá prostřednictvím klávesnice a myši. Mnohdy je však implementována i podpora dalších zařízení, například gamepadu, joysticku atd. Způsob implementace a vzhled grafického uživatelského rozhraní je závislý na použité knihovně pro tvorbu GUI a na cílové platformě, na které je aplikace s GUI spuštěna.

Komunikace s rozhraním je postavena na principu generování událostí (anglicky „events“). To znamená, že po spuštění aplikace s GUI dojde k inicializaci uživatelského rozhraní. Mechanismus zajišťující obsluhu rozhraní potom čeká, až dojde k vyvolání události. Poté je událost vyhodnocena a mechanismus opět čeká na další událost. K jeho ukončení dochází současně s ukončením aplikace (v opačném případě by došlo k havárii aplikace). Příklad vzhledu uživatelského rozhraní znázorňuje Obr. 6.



Obr. 6. Příklad vzhledu grafického uživatelského rozhraní

4.1.5 Multimediální rozhraní

Jedná se spíše o přídatnou část uživatelského rozhraní, která do procesu komunikace s počítačem umožňuje zapojit například hlas, gestikulaci nebo dotek. Hlavně způsob ovládání přístroje, postavený na základě snímání pohybů lidského těla, doznal za posledních několik let velmi znatelných pokroků, a to hlavně v herním odvětví

(Nintendo Wii, PlayStation Move, zejména pak Microsoft Kinect, který umožňuje ovládat přístroj pomocí pohybů celého lidského těla a to pouze za pomoci speciální kamery, tzn. bez použití jakýchkoliv dalších zařízení připnutých nebo držených uživatelem).

II. PRAKTICKÁ ČÁST

5 PRAKTICKÝ ÚVOD

Tato kapitola se zabývá souhrnem základních informací spojených s tvorbou programové části diplomové práce.

5.1 Zavedení pojmů

V následujícím textu bude často zmiňována operace se soubory i složkami, přičemž informace, zda-li se jedná o soubor nebo o složku, bude v danou chvíli čistě irelevantní. Také se zde bude vyskytovat situace, kdy bude teprve docházet ke zjištění, zda-li se jedná o soubor nebo o složku. V obou těchto případech si proto zavedme univerzální pojem „*položka*“, který bude v danou chvíli definovat buďto soubor nebo složku (vždy alespoň právě jednu z těchto možností).

Dále je vhodné zmínit, že veškeré číselné rozsahy a hodnoty uváděné v bajtech, jsou uváděny pro programovací jazyk C++⁴. Z pohledu výše zmíněného programovacího jazyka se tyto údaje mohou lišit v závislosti na platformě⁵, na které dochází k překladu aplikace (vzhledem k systému, na němž byla programová část diplomové práce vytvářena, uvažujme 32-bitovou implementaci). Jedná se o snadno přehlédnutelnou skutečnost, která však může způsobit nemalé komplikace a případné havárie aplikace. Typickým příkladem výše popisované problematiky může být například datový typ Integer, jehož datový rozsah je 2^{16} (u 16-ti bytových systémů) nebo 2^{32} (u 32-bitových systémů).

Velmi významnou úlohu má v implementaci archivátoru údaj „identifikační číslo“ (zkráceně „ID“). Jedná se o číslo datového typu Integer, které jednoznačně identifikuje jednotlivé položky obsažené v každé položce typu soubor hlavního (kořenového) vektoru. Podstata identifikačního čísla bude vysvětlena v kapitole, která se zabývá se stromovou strukturou (kapitola 8.3).

V diplomové práci je používán převod, kdy 1 kB je roven 1024 bajtům. Tento převod se dá teoreticky považovat za nesprávný, jelikož, dle tabulky SI jednotek, by měla pro tento

⁴ Programovací jazyk z rodiny C/C++/Objective-C.

⁵ Platforma = operační systém + hardware.

převod být použita jednotka „KiB“ (Kibibajt) namísto použitého „kB“. Tato skutečnost byla ověřena na operačních systémech Microsoft Windows a Linux. I na těchto systémech je používán převod $1 \text{ kB} = 1024 \text{ B}$ místo převodu $1 \text{ kB} = 1000 \text{ B}$. Z tohoto důvodu bylo v aplikaci využito stejného převodního vztahu jako u výše zmiňovaných operačních systémů.

5.2 Požadavky na aplikaci

„Návrh každého projektu je spjat se zjišťováním co největšího počtu informací o problematice, kterou se zabývá. Projekt postavený na základě smyšlených údajů a/nebo nedostatečných znalostí je v praxi nepoužitelný a tím pádem již předem odsouzen k neúspěchu.“[3] Samozřejmou součástí každého aplikačního návrhu je zjištění požadavků, které jsou kladeny na aplikaci daného typu a současně také stanovení cílů, kterých má být při vývoji aplikace dosaženo.

Pro aplikaci zabývající se kompresí a dekompresí dat (dále jen „archivátor“) lze uvažovat následující požadavky:

- **Rychlost** – co nejkratší časová náročnost kompresního/dekompresního procesu.
- **Kvalita** – dosažení co nejlepší komprese, tzn. co nejnížší hodnoty získané poměrem mezi komprimovanou a nekomprimovanou podobou souboru.
- **Efektivita procesu** – kompresní proces by měl v co nejkratším čase dosáhnout co nejlepší komprese, tzn. co nejvyšší hodnoty získané poměrem mezi poměrem rozdílu nekomprimované verze vůči komprimované verzi souboru a časem, za který bylo komprese dosaženo.
- **Spolehlivost aplikace** – aplikace musí fungovat bez různých chyb a nesrovnalostí, které by znepříjemňovaly práci uživatele nebo dokonce způsobovaly pády aplikace.
- **Spolehlivost komprese** – soubory zkomprimované archivátorem musí jít zpětně dekomprimovat do takové podoby, aby byly naprosto identické s originály.
- **Úplnost** – při procesu archivace musí být archivovány všechny položky, které se tohoto procesu účastní (tzn. včetně prázdných souborů, prázdných složek, vnořených složek a souborů v těchto složkách, skrytých souborů atd.).

- **Blokace akcí** – uživatel by nemělo být umožněno vyvolávat formuláře nebo spouštět akce, které jsou v dané situaci irelevantní. Například by nemělo být uživateli umožněno vyvolat „extrakci“⁶ souborů z archivu, není-li žádný archiv načten.
- **Minimální nadbytečnost** – archivátorem vytvořený archiv by měl mít co nejmenší velikost = optimalizovaná struktura obsahující pouze důležité informace potřebné pro dekompresi archivu.
- **Jednoduchost a přehlednost** – rozhraní aplikace by mělo být co nejvíce uživatelsky přívětivé a přehledné, práce s aplikací by měla být co nejjednodušší. Varovná hlášení by měla uživatele navést správným směrem vedoucím k vyřešení problému/nedostatku.
- **Informace o průběhu** – v případě, že v aplikaci dochází k časově náročným operacím (např. komprese a dekomprese), by měl být uživatel o jejich průběhu pravidelně informován.
- **Kompatibilita** – aplikace by měla být použitelná na co největším počtu různých platforem.
- **Hospodárnost** – aplikace by neměla alokovat zbytečně velké paměťové prostory. Naopak by měla již za běhu pravidelně uvolňovat paměťový prostor, který již není potřebný a předcházet tak únikům paměti (anglicky „*Memory Leaks*“), tzn. vzniku oblastí v paměti, které zůstávají alokovány i po ukončení/pádu aplikace.

5.3 Použité technologie

Program byl vytvořen v programovacím jazyce C++, přičemž pro tvorbu grafického uživatelského rozhraní byla využita multiplatformní knihovna Qt verze 4.7.0 (32 bit). Volba tohoto typu uživatelského rozhraní byla ovlivněna operačním systémem, pro nějž byla vyvíjena a očekáváním uživatelů na způsob komunikace mezi nimi a aplikací pro tento operační systém vytvořenou.

Aplikace byla vytvořena ve vývojovém prostředí Qt Creator verze 2.0.1.

⁶ Operaci lze jinými slovy také nazvat „rozbalením“ nebo „dekompresí“.

Na základě výsledků analýzy požadavků kladených na aplikace typu „archivátor“ byla zvolena binární podoba výstupního souboru (archivu).

6 NÁVRH

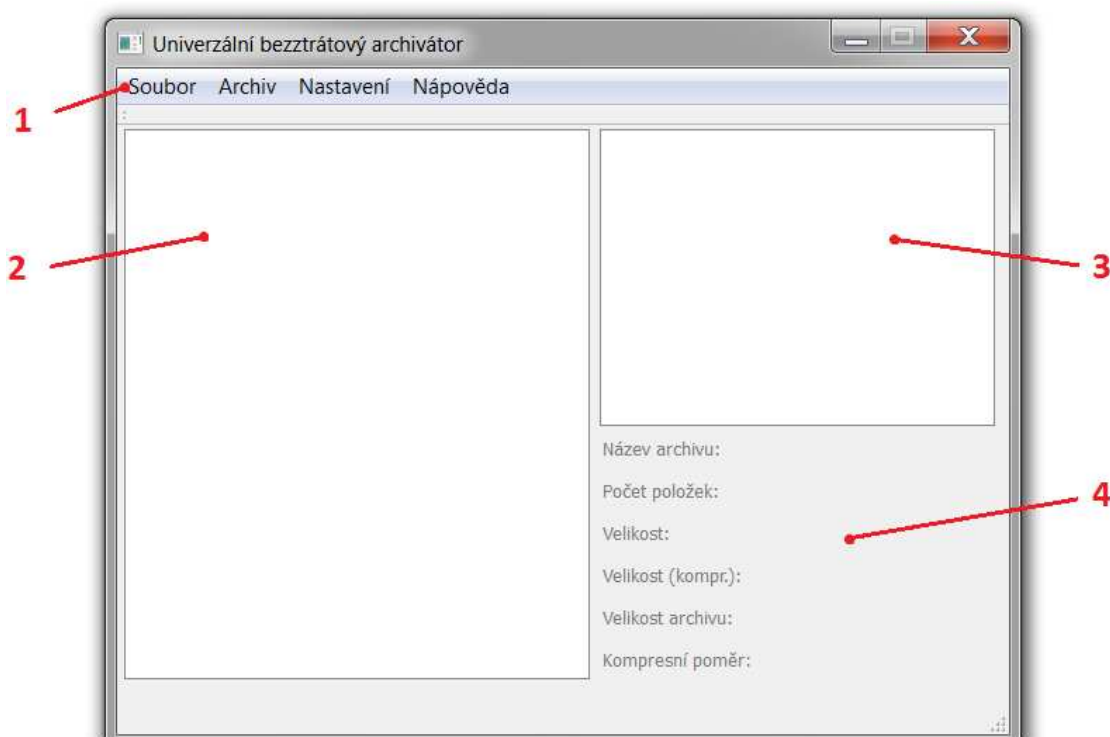
Samotný návrh struktury Univerzálního bezztrátového archivátoru sestává z několika dílčích částí. První část přiblíží návrh grafického uživatelského rozhraní. Současně zde bude vysvětlen význam jednotlivých dialogových oken aplikace a to včetně jejich využití. Samozřejmostí kapitoly je popis datových struktur využívaných v aplikaci. Poslední část je věnována popisu návrhu struktury archivu, který lze prostřednictvím aplikace vytvořit.

6.1 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní aplikace sestává z osmi oddělených dialogových oken. Každé z těchto oken má svoje specifické vlastnosti a využití, přičemž výsledkem jejich spojení je plnohodnotný nástroj pro komunikaci uživatele s aplikací. V rámci této kapitoly budou jednotlivá okna (a možnosti jejich využití) podrobněji popsána:

6.1.1 Hlavní okno aplikace

Hlavní dialogové okno (viz Obr. 7.) se zobrazí bezprostředně po spuštění aplikace. Umožňuje provádět základní operace a současně propojuje ostatní dialogová okna do jednoho funkčního celku.

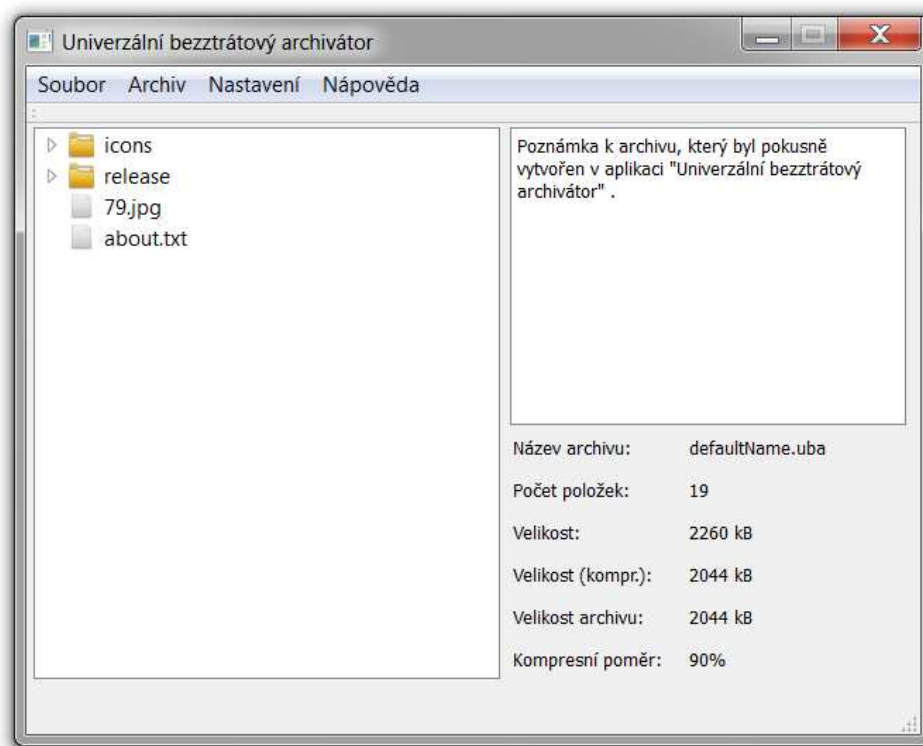


Obr. 7. Popis hlavního okna aplikace

- 1) Hlavní menu dialogového okna obsahuje základní funkce aplikace.
- 2) Komponenta zobrazující stromovou strukturu načtených položek nebo položek archivovaných v načteném archivu.
- 3) Needitovatelné textové pole pro zobrazení poznámky v archivu (je-li načten).
- 4) Základní údaje o položkách načtených v aplikaci. Tyto údaje se doplňují v závislosti na aktuální situaci. V případě vytváření nového archivu se zobrazují jen informace o počtu položek a jejich velikosti. Je-li načten již vytvořený archiv, jsou zobrazeny všechny údaje:
 - a) **Název archivu** – název načteného archivu
 - b) **Počet položek** – informace o celkovém počtu načtených položek. Po najetí myši na hodnotu tohoto údaje dojde k zobrazení popisu komponenty⁷ (dále jen „*tooltip*“) s rozdělením tohoto údaje podle typu položky, tzn. počet načtených souborů a počet načtených složek.
 - c) **Velikost** – původní celková velikost načtených položek, tzn. velikost položek před kompresí; hodnota udávaná v kB, v tooltipu je shodný údaj uveden v bajtech
 - d) **Velikost (kompr.)** – celková velikost položek po kompresním procesu; hodnota udávaná v kB, v tooltipu je shodný údaj uveden v bajtech
 - e) **Velikost archivu** – celková velikost archivu, tzn. celková velikost položek v archivu plus velikost všech přídatných informací, které se v archivu nacházejí; hodnota udávaná v kB, v tooltipu je shodný údaj uveden v bajtech
 - f) **Kompresní poměr** – procentuální velikost archivu v poměru k velikosti položek před kompresí

⁷ Anglicky „*tooltip*“; uživatelem needitovatelné textové pole; je standardně neviditelné a zobrazí se jen v případě, že uživatel určitou dobu ponechá kurzor myši na nějakou z komponent, která tooltip využívá.

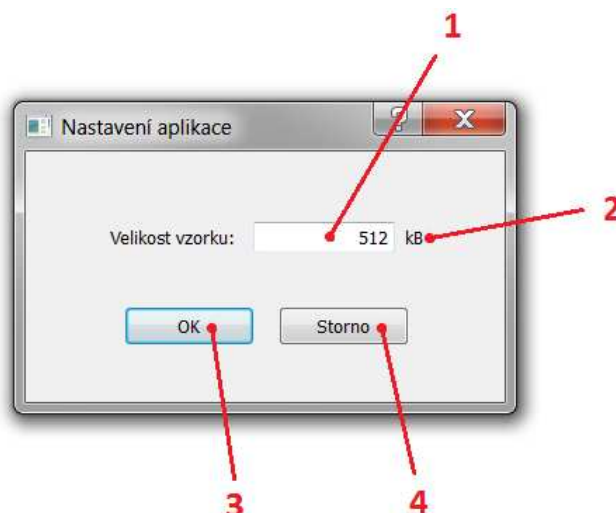
Po načtení již vytvořeného archivu může hlavní okno aplikace vypadat tak, jak je uvedeno na Obr. 8.



Obr. 8. Vzhled hlavního okna aplikace po načtení archivu

6.1.2 Nastavení aplikace

Dialogové okno, sloužící pro nastavení aplikace (viz Obr. 9.), se spouští prostřednictvím menu hlavního okna aplikace (Nastavení → Aplikace). Prostřednictvím tohoto dialogu lze nastavit jediný modifikovatelný parametr aplikace, tzn. velikost vzorku (bližší popis viz kapitola 8.2).



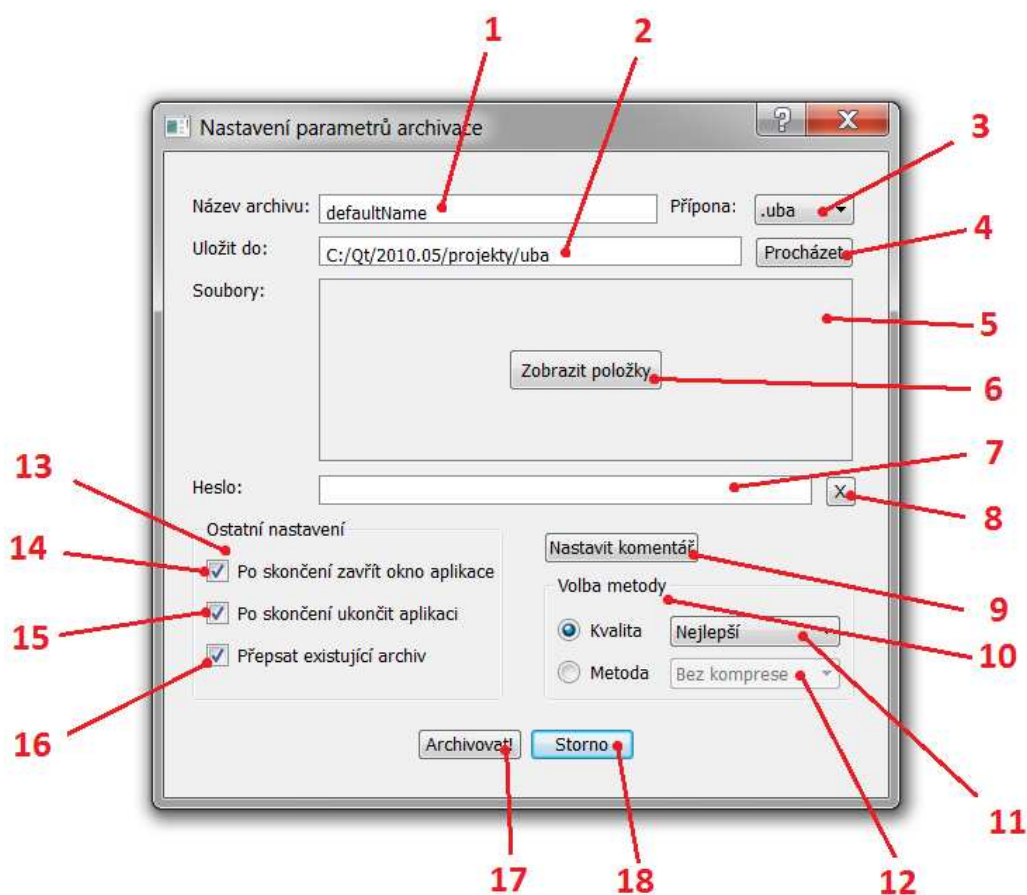
Obr. 9. Popis dialogu pro nastavení aplikace

- 1) Textové pole pro zadání číselné hodnoty udávající požadovanou velikost (v kB); tooltip tohoto pole upozorňuje na převodní poměr mezi kB a B⁸ a doporučenou hodnotu, která je rovna hodnotě defaultní, tzn. 512 kB
- 2) Informativní text upozorňující uživatele na jednotky, ve kterých má být daná velikost zadávána
- 3) Tlačítko pro potvrzení změn; je-li aktivováno, dojde ke zpracování zadané hodnoty a v případě potřeby je původní hodnota změněna na hodnotu aktuálně zadanou
- 4) Tlačítko pro zavření tohoto dialogu a pro zrušení případně provedených změn

⁸ 1 kB = 1024 B, bliže viz kapitola 5.1.

6.1.3 Nastavení parametrů archivace

Dialogové okno (viz Obr. 10.) se spouští prostřednictvím menu hlavního okna aplikace (Archiv → Vytvořit archiv). Slouží pro nastavení parametrů, které budou následně ovlivňovat archivační proces. Současně se přes toto dialogové okno spouští samotný proces archivace.



Obr. 10. Popis dialogu pro nastavení parametrů archivace

- 1) Textové pole pro zadání názvu archivu, který má být vytvořen. Název archivu je implicitně přednastaven v závislosti na počtu kořenových položek, které se mají v archivu nacházet:
 - je-li v aplikaci načtena pouze jedna kořenová položka, je název archivu odvozen od jejího názvu (v případě složky je název archivu totožný s názvem této složky; pokud je jedinou kořenovou položkou soubor, je název archivu totožný s názvem

tohoto souboru bez přípony, tzn., že pro soubor s názvem „dokument.txt“ je název archivu stanoven na „dokument“⁹)

- je-li v aplikaci načteno více kořenových položek, je název archivu stanoven neutrálně na „archiv“

Přednastavená hodnota slouží spíše pro usnadnění práce s archivátorem¹⁰, lze ji však libovolně změnit.

- 2) Textové pole určené pouze pro čtení. Udává, kde bude archiv vytvořen.
- 3) Seznam s řádkem sloužící pro volbu přípony archivu. Komponenta je předpřipravena pro případné další rozšíření, zatím však nabízí pouze příponu „uba“.
- 4) Pomocí tohoto tlačítka lze změnit cestu, která udává, kde bude archiv vytvořen (viz komponenta číslo 2 tohoto dialogového okna).
- 5) Neditovatelné textové pole pro zobrazení všech položek, které mají být v archivu obsaženy. Z důvodu časové náročnosti zobrazení všech položek a s přihlédnutím ke skutečnosti, že tento údaj je spíše údajem doplňujícím, je pole bezprostředně po zobrazení tohoto dialogu prázdné.
- 6) Tlačítko, které slouží pro naplnění textového pole (komponenta číslo 5 tohoto dialogového okna) patřičnými údaji, tzn. seznamem načtených položek. Po stisknutí tohoto tlačítka dojde ke změně jeho textu ze „Zobrazit položky“ na „Čekejte...“. Dochází k procesu načítání položek, na jehož konci dojde k naplnění výše zmiňovaného textového pole. Aktuálně popisované tlačítko přestane být, z pohledu další práce s aplikací, po naplnění textového pole důležité, a je proto skryto.
- 7) Textové pole pro zadání hesla, kterým bude následně vytvořený archiv zabezpečen.
- 8) Tlačítko pro zrušení obsahu textového pole pro zadání hesla (komponenta číslo 7 tohoto dialogového okna).

⁹ Plus přípona „uba“.

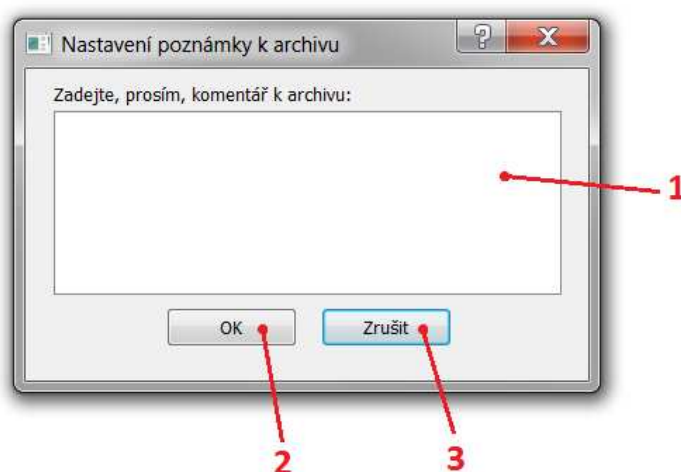
¹⁰ Název archivu je povinným údajem a uživatel ho, vzhledem k implicitně nastavené hodnotě, není nucen vyplňovat.

- 9) Tlačítko vyvolá dialogové okno pro nastavení poznámky k archivu (dále viz kapitola 6.1.4).
- 10) Přepínač, který umožňuje uživateli volit mezi dynamickou nebo statickou volbou kompresní metody. Dynamická volba (možnost „*Kritérium*“ tohoto dialogového okna) probíhá na základě výsledků testů prováděných během procesu archivace na vzorku archivovaného souboru. Při statické volbě (možnost „*Metoda*“ tohoto dialogového okna) je uživatelem vybrána konkrétní kompresní metoda, která bude použita při kompresi všech archivovaných souborů.
- 11) Při dynamické volbě kompresní metody je po uživateli požadována volba kritéria, na základě kterého bude nejvhodnější kompresní metoda určována. Lze volit mezi těmito kritérii:
 - a) **Kvalita** – kompresní metoda je volena na základě kvality komprese, tzn., na kolik procent původní velikosti dokáže metoda soubor zkomprimovat
 - b) **Kvalita (vše)** – stejné kritérium jako u předchozí možnosti. Rozdíl je pouze ve skutečnosti, že není-li pro daný soubor efektivní žádná kompresní metoda (v případě, že zkomprimovaný soubor dosahuje větší velikosti než jeho nezkomprimovaná podoba), je tento soubor do archivu vložen v nezkomprimované podobě.
 - c) **Efektivita** – volba probíhá na základě poměru mezi počtem procent, o kolik je metoda schopna snížit velikost souboru, a časem, za který je této komprese dosaženo
 - d) **Efektivita (vše)** – stejné kritérium volby kompresní metody jako u předchozí možnosti. Uvažuje se však také metoda „bez komprese“, což znamená, že obsah souboru může být do archivu překopírován v nezkomprimované podobě.
 - e) **Rychlost** – vhodná metoda je volena na základě rychlosti komprese
- 12) Statická volba kompresní metody – uživatel volí metodu z nabídky všech implementovaných kompresních metod.
- 13) Skupina zastřešuje rozšiřující nastavení kompresního procesu.
- 14) Je-li tato možnost zaškrtnuta, je archivační/extrakční dialog (viz kapitola 6.1.6), po ukončení kompresního procesu, automaticky ukončen.

- 15) Je-li spolu s tlačítkem 14 tohoto dialogového okna zaškrtnuto i toto tlačítko, je po ukončení kompresního procesu automaticky ukončena celá aplikace.
- 16) Zaškrtnutím této možnosti se předejde dotazům na přepsání již existujícího archivu se shodným jménem.
- 17) Tlačítko spouštějící proces archivace.
- 18) Tlačítko ukončí aktuální dialogové okno a vrátí uživatele zpět do hlavního okna aplikace.

6.1.4 Nastavení poznámky k archivu

Pro větší přehlednost¹¹ dialogu, sloužícího k nastavení parametrů archivačního procesu, bylo textové pole pro zadání poznámky k archivu odděleno od ostatních komponent tohoto nastavení (viz Obr. 11.).



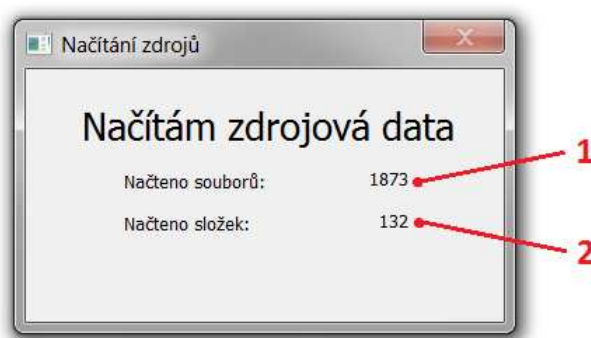
Obr. 11. Popis dialogu pro nastavení poznámky

- 1) Textové pole pro zadání komentáře k archivu.
- 2) Tlačítko pro akceptování provedených změn.
- 3) Tlačítko pro zrušení provedených změn.

¹¹ Vycházelo se z předpokladu, že poznámka k archivu není využívána každým uživatelem. Dalším předpokladem pro oddělení pole pro zadání poznámky bylo, že je-li tato možnost využita, může poznámka nabývat větších rozměrů.

6.1.5 Načítání zdrojů

Při práci s aplikací dochází k časově náročnějším operacím, při nichž by měl být uživatel o jejich průběhu pravidelně informován. K takovým operacím patří načítání archivu s velkým množstvím archivovaných položek a načítání velkého množství zdrojů (například načtení složky s „bohatou stromovou strukturou“¹²) do aplikace. Uživatel je o průběhu těchto operací informován prostřednictvím dialogového okna (viz Obr. 12.).



Obr. 12. Popis dialogu pro načítání zdrojových dat

- 1) Průběžně aktualizovaný údaj o počtu načtených souborů.
- 2) Průběžně aktualizovaný údaj o počtu načtených složek.

6.1.6 Archivační/extrakční dialogové okno

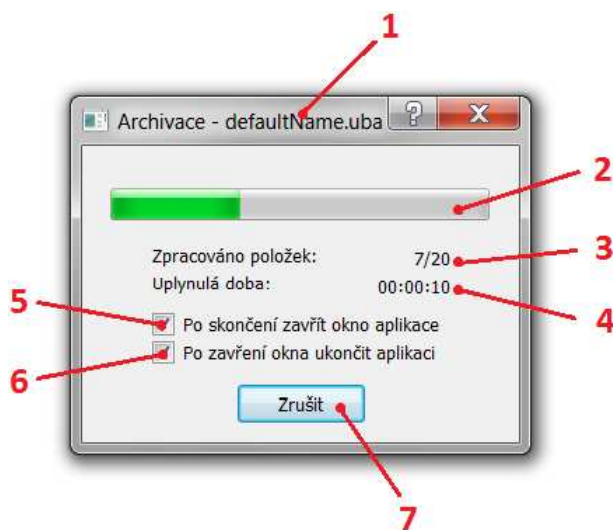
Časově nejnáročnější operací při práci s archivátorem je samotný archivační/extrakční proces. Ten se spouští přes dialogové okno pro nastavení parametrů archivace¹³ (viz kapitola 6.1.3) a nebo prostřednictvím nabídky „Archiv → Extrahovat (do)“¹⁴ v hlavním oknu aplikace. Doba jeho trvání je závislá na mnoha různých parametrech (výkon a vytížení počítače, počet a velikost archivovaných souborů, zvolená kompresní

¹² Velké množství hierarchicky uspořádaných souborů a složek.

¹³ Dojde ke spuštění procesu archivace.

¹⁴ Dojde ke spuštění procesu extrakce.

metoda,...), a proto by měl být uživatel o jeho průběhu pravidelně informován. K tomu slouží dialog uvedený na Obr. 13. Dialogové okno je pro archivaci a extrakci shodný.

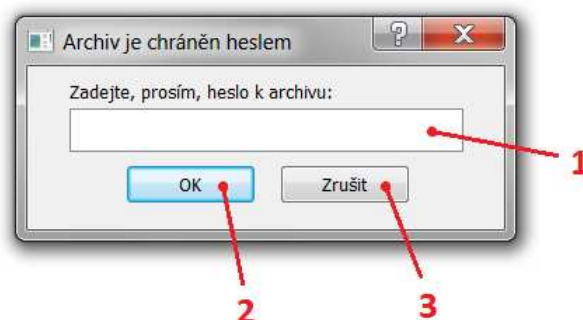


Obr. 13. Popis argivačního/extracního dialogu

- 1) Titulek dialogového okna se mění v závislosti na prováděné operaci (archive/extrakce) a zpracovávaném archivu, jehož název je v titulku dialogového okna také uveden.
- 2) Grafické znázornění průběhu prováděného procesu. Zde je znázorněn poměr mezi zpracovanými a nezpracovanými položkami.
- 3) Textovým způsobem vyjádřený průběh prováděného procesu. Před lomítkem je uveden počet zpracovaných položek, za lomítkem se nachází celkový počet položek, které mají být v procesu zpracovány.
- 4) Měření času uplynulého od spuštění procesu.
- 5) Zaškrťovací políčko, které stanovuje, zda-li má, po skončení probíhajícího procesu, dojít k automatickému zavření tohoto dialogového okna.
- 6) Je-li s tlačítkem číslo 5 tohoto dialogového okna zaškrtnuto i toto políčko, je po dokončení archivačního/extrakčního procesu ukončena celá aplikace.
- 7) Tlačítko pro ukončení archivačního/extrakčního procesu. Po předčasném ukončení probíhajícího procesu je tento proces neúspěšný. Je-li však proces, o jehož průběhu dialogové okno informuje, úspěšně ukončen, je text tlačítka změněn na „Hotovo“.

6.1.7 Zadání hesla k archivu

Je-li prostřednictvím aplikace načten archiv zabezpečený heslem, dojde k automatickému zobrazení dialogového okna, které uživatele na tuto skutečnost upozorní a vyzve ho k zadání tohoto hesla (viz Obr. 14.).

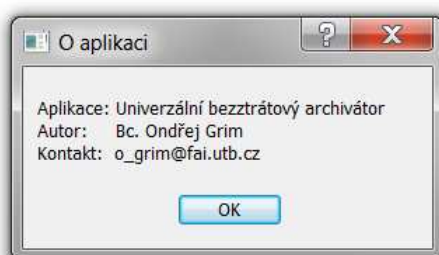


Obr. 14. Popis dialogu pro zadání hesla

- 1) Textové pole pro zadání hesla.
- 2) Tlačítko pro potvrzení zadané hodnoty. Pokud je zadaná hodnota nesprávná, je uživatel na tuto skutečnost upozorněn varovnou zprávou. Toto dialogové okno zůstane dále aktivním až do zadání správného hesla.
- 3) Tlačítko pro zavření tohoto dialogového okna. Proces načítání archivu je bezprostředně po stisknutí tlačítka ukončen.

6.1.8 O aplikaci

Spouští se z menu hlavního okna aplikace (Nápověda → O aplikaci). Jednoduché dialogové okno, které slouží pouze k zobrazení základních údajů o této aplikaci (viz Obr. 15.). Dané okno má pouze informativní charakter, neposkytuje žádnou využitelnou funkcionalitu.



Obr. 15. Popis dialogu
s informacemi o aplikaci

6.2 Datové struktury

V rámci implementačního postupu došlo k vytvoření třech datových struktur, kterých se s výhodou využívá v různých částech programu:

6.2.1 Struktura „Folder“:

Slouží k uchování kořenových položek. Jednotlivé proměnné této datové struktury jsou popsány v Tab. 1.

Proměnná struktury	Datový typ	Popis
Path	QString	cesta k položce
Name	QString	jméno položky
isDirectory	bool	typ položky („TRUE“ = složka, „FALSE“ = soubor)
fileSize	long long	velikost souboru
Files	QVector<File>	vektor podřízených položek
posAtFile	long long	pozice v souboru (význam pouze při dekompresi archivu, do archivu se neukládá; udává pozici, na které se v archivu nachází začátek obsahu souboru)
Method	unsigned short	zvolená metoda komprese

Tab. 1. Popis proměnných datové struktury „Folder“

6.2.2 Struktura „File“:

Slouží k uchování položek podřazených kořenovým položkám. Jednotlivé proměnné této datové struktury jsou popsány v Tab. 2.

Proměnná struktury	Datový typ	Popis
id	int	identifikační číslo položky
name	QString	jméno položky
path	QString	cesta k položce
parent	int	identifikační číslo nadřazené položky (složky, ve které se tato položka přímo nachází)
fileSize	long long	velikost souboru
isDirectory	bool	typ položky („TRUE“ = složka, „FALSE“ = soubor)
posAtFile	long long	pozice v souboru (význam pouze při dekompresi archivu, do archivu se neukládá; udává pozici, na které se v archivu nachází začátek obsahu souboru)
method	unsigned short	zvolená metoda komprese

Tab. 2. Popis proměnných datové struktury „File“

Způsob pojmenování struktur slovy „Folder“ (česky „složka“) a „File“ (česky „soubor“) je sice poněkud zavádějící. Ale dané názvy byly zvoleny s přihlédnutím ke vzájemnému vztahu těchto struktur. Znamená to, že nadřazená struktura „Folder“ se nachází v pomyslném hlavním (kořenovém) adresáři a zapouzdřuje v sobě všechny položky v ní obsažené bez rozdílu jejich typu (vektor struktur „File“).

Funkčnost aplikace je tedy primárně postavena na „hlavním“ vektoru datových struktur typu „Folder“, přičemž každá položka tohoto vektoru obsahuje vektor datových struktur typu „File“.

6.2.3 Struktura „MethodChoose“:

Využívá se při archivaci (konkrétně při volbě kompresní metody), kde vektor struktur typu „MethodChoose“ uchovává výsledky jednotlivých kompresních metod. Jednotlivé proměnné této datové struktury jsou popsány v Tab. 3.

Proměnná struktury	Datový typ	Popis
fileSizeAfter	long long	velikost vstupních dat po kompresním procesu; za jistých okolností udává velikost proměnné „fileContentAfter“ (dále viz kapitola 8.2.4)
ratio	long	kompresní poměr mezi vstupním a výstupním obsahem (v procentech); nižší je lepší; matematicky: $\left(\frac{VELIKOST_VSTUPU}{fileSizeAfter} \right) * 100$, kde <i>VELIKOST_VSTUPU</i> je velikost dat vstupujících do kompresního procesu
timeDuration	int	doba trvání komprese; nižší je lepší
ratioWithTime	float	efektivita komprese za jednotku času; vyšší je lepší; matematicky: $\frac{100 - \left(\left(\frac{VELIKOST_VSTUPU}{fileSizeAfter} \right) * 100 \right)}{timeDuration}$, kde <i>VELIKOST_VSTUPU</i> je velikost dat vstupujících do kompresního procesu; vzorec lze zjednodušit na: $\frac{100 - ratio}{timeDuration}$
fileContentAfter	char*	komprimovaný obsah souboru; využívá se jen v případě, že soubor splňuje jistá kritéria (dále viz kapitola 8.2.4)
wholeFile	bool	údaj o tom, zda je proměnná „fileContentAfter“ naplněna vhodným obsahem (viz kapitola 8.2.4)

Tab. 3. Popis proměnných datové struktury „MethodChoose“

6.3 Archiv

Jak je již zmíněno v kapitole 5.3, která pojednává o použitých technologiích, je archiv binárním souborem. Tento typ souboru je použit z důvodu větší rychlosti operací se souborem a úspory místa (detailněji vysvětleno v kapitole 3.3).

6.3.1 Obecné vlastnosti

Pro odlišení archivů vytvořených touto aplikací od ostatních typů souborů byla zvolena přípona „uba“ (zkratka výrazu „univerzální bezztrátový archivátor“). Přípona byla volena tak, aby se předešlo víceznačnosti interpretace této přípony jak samotným uživatelem, tak i operačním systémem (ten má k různým příponám asociovány různé programy) a nedocházelo tak k nežádoucím záměnám s jinými typy souborů. Vzhledem ke skutečnosti, že v době, kdy vznikala tato diplomová práce, nebyla zjištěna existence žádného úřadu, který by normalizoval systém přípon souborů, není zcela vyloučeno, že tutéž příponu v současnosti nevyužívá vůbec žádný program nebo že v budoucnosti nevznikne program, který by tuto příponu využíval.

6.3.2 Obsah archivu

Aplikací vytvořený archiv v sobě musí zahrnovat:

- obsahy všech souborů, které se v něm nacházejí
- informace o jednotlivých položkách archivu, na základě kterých lze procesem extrakce vytvořit soubory/složky, které jsou, při porovnání s jejich originálními verzemi (předlohami), zcela identické
- informace o stromové struktuře, které umožňují zrekonstruovat stromovou strukturu archivovaných položek
- další záznamy, které nejsou pro rekonstrukci archivovaných dat podstatné, ale svým využitím podporují rozšířené funkce archivátoru, např. komentář k archivu, heslo atd.
- údaje o velikostech jednotlivých záznamů

Je vhodné se u poslední odrážky krátce pozastavit a vysvětlit si význam zmiňovaných záznamů. Co je to vlastně obsah binárního souboru pro počítač? Je to řada po sobě jdoucích bajtů různých hodnot bez logického významu. Počítač strukturu obsahu nerozumí, neví, že např. v prvních dvou bajtech je uložena hodnota typu „unsigned long

integer¹⁵“, za níž následuje jeden bajt s hodnotou např. typu „char“ (znak). Způsob, jakým má počítač význam jednotlivých bajtů chápat, mu musí být explicitně zadán, a to prostřednictvím aplikace (resp. v konkrétní funkci, která zpracování obsahu archivu řeší).

Informace uložené v archivu lze rozdělit na „obecná data“ a „data položek“.

Obecná data:

Záznamy podstatné pouze pro archiv jako takový – jedná se o záznamy, které nejsou (s výjimkou údaje o počtu souborů v archivu) klíčové pro správnou funkcionalitu. S trochou nadsázky by se o nich dalo mluvit jako o záznamech, které podporují „nadstandardní“ funkce aplikace. Obecná data jsou popsána v Tab. 4.

Informace	Datový typ	Velikost	Popis
Zadáno heslo	boolean	1 bajt	udává, zda-li je zadáno heslo k archivu (hodnota „true“) nebo není (hodnota „false“)
Délka hesla	integer	4 bajty	délka zadaného hesla
Text hesla	char*	N * 1 bajt	samotná textová hodnota hesla; N je rovno hodnotě „Délka komentáře“
Zadán komentář	boolean	1 bajt	udává, zda-li je zadán komentář k archivu (hodnota „true“) nebo není (hodnota „false“)
Délka komentáře	integer	4 bajty	délka zadaného komentáře k archivu
Text komentáře	char*	N * 1 bajt	samotná textová hodnota komentáře; N je rovno hodnotě „Délka komentáře“
Velikost dat před kompresí	long long	8 bajtů	původní velikost všech souborů obsažených v archivu
Počet souborů	integer	4 bajty	počet souborů v archivu

Tab. 4. Popis obecných dat

¹⁵ Uvažujeme-li 32 bitový systém, jde o rozsah hodnot $0 - 2^{32}$ neboli o interval $\langle 0; 4\,294\,967\,296 \rangle$.

Data položek:

Záznamy jsou spjaty s jednotlivými položkami, které jsou v archivu obsaženy. Opomeneme-li jistou zavedenou optimalizaci (dále viz kapitola 6.3.3), lze konstatovat, že tyto záznamy se v archivu objevují N -krát, kde N je počet položek obsažených v archivu. Data položek jsou popsána v Tab. 5.

Informace	Datový typ	Velikost	Popis
ID položky	Integer	4 bajty	identifikační číslo aktuální položky
ID rodiče	integer	4 bajty	identifikační číslo nadřazené složky, ve které se položka nachází
Délka názvu	integer	4 bajty	délka názvu položky (u souborů včetně přípony)
Název položky	char*	$N * 1$ bajt	název položky (u souborů včetně přípony); N je rovno hodnotě „Délka položky“
Typ položky	boolean	1 bajt	udává, zda-li se jedná o položku (hodnota „true“) nebo soubor (hodnota „false“)
Kompresní metoda	unsigned short	2 bajty	pevně dané číslo kompresní metody, která byla zvolena pro komprimaci souboru
Velikost dat	long long	8 bajtů	velikost obsahu souboru (po zvolené kompresi)
Obsah souboru	char*	$N * 1$ bajt	samotný obsah souboru (po zvolené kompresi); N je rovno hodnotě „Velikost dat“

Tab. 5. Popis dat položek

6.3.3 Optimalizace dat v archivu

S přihlédnutím k jednomu z požadavků na aplikaci typu „archivátor“, konkrétně na „velikost archivu“, dochází při zaznamenávání jednotlivých informací k jistým drobným optimalizacím.

První optimalizační stupeň nastává u kořenových položek (položky s ID 0, bližší vysvětlení viz kapitola 8.3), kde se neukládají identifikační čísla jejich rodičů, tzn. jim nadřazených složek. Pro tyto složky totiž žádná nadřazená složka v podstatě neexistuje. Vhodným ošetřením funkcí, které vytvářejí/zpracovávají archiv, tak lze ušetřit 4 bajty na každou kořenovou položku.

Složka z pohledu souborového systému může pouze obsahovat předem nespécifikované množství (včetně nuly) dalších souborů a složek, které jsou jí „podřízené“. Složka jako taková však nenese žádná data srovnatelná s daty v tělech souborů. Lze tedy předem konstatovat, že velikost dat v těle složky je rovna nule. Vhodnou optimalizací funkcí pracujících s archivy tak lze ušetřit 8 bajtů, jelikož není nutné ukládat informaci o velikosti prázdné složky, plus 1 bajt v případě, že by byl obsah složky v archivu nahrazen například prázdným nebo ukončovacím znakem. U složek nelze komprimovat žádný obsah. Z toho vyplývá, že nedochází k uložení informace o zvolené kompresní metodě a tak k úspoře dalších 2 bajtů. Znamená to tak celkovou úsporu 10 – 11 bajtů na přídatných informacích na položku typu „složka“. Stejná optimalizace platí také u souborů, které neobsahují žádná data, např. txt soubor, který byl pouze vytvořen, ale nebylo do něj nic zaznamenáno. Tím dochází k další úspoře 10 – 11 bajtů na přídatných informacích na položku typu „prázdný soubor“. Úsporný mechanismus je založen na informaci, kterou by bylo vhodné uchovávat i v případě jeho absence, tj. informace udávající, zda-li je položka složkou nebo souborem.

V neposlední řadě došlo k jisté optimalizaci obecných informací obsažených v archivu. Teoreticky tak lze ušetřit 4 bajty za absenci údaje o délce hesla a 0 – 1 bajt za absenci textové hodnoty hesla (opět v závislosti na implementaci – zda-li nedochází k nahrazení textové hodnoty prázdným nebo ukončovacím znakem). Stejný mechanismus platí také pro komentář. Teoreticky tak lze ušetřit 2 krát 4 až 5 bajtů, tzn. úspora 8 – 10 bajtů.

Vzhledem k faktu, že zde muselo dojít k ukládání nové informace (údaj o zadání/nezadání hesla nebo komentáře), jejíž užitek je při dalších operacích s archivem nulový, je vhodné tuto nadbytečnou informaci o velikosti 2 krát 1 bajt (1 bajt pro heslo a 1 bajt pro komentář) zohlednit při výpočtech kvality optimalizace. Reálná úspora je tedy o 2 krát 1 bajt menší, tzn. pouze 6 – 8 bajtů, a to pouze za předpokladu, že nedošlo k zadání hesla ani komentáře. V opačném případě má totiž optimalizační mechanismus negativní účinky, tedy nárůst velikosti obecných záznamů o 1 (v případě zadání pouze hesla nebo komentáře) až 2 (v případě zadání hesla a současně i komentáře) bajty.

7 OPERACE S ARCHIVEM

Nejvýznamnější kapitolou zabývající se praktickou částí jsou operace s archivem od samotného načtení zdrojového souboru či složky, přes její kompresi, až po načtení již vytvořeného archivu a dekompresi položek v něm obsažených. Proto je oblastí, která se zabývá touto problematikou, věnována jedna celá hlavní kapitola.

7.1 Položky v aplikaci

Základním předpokladem pro pochopení následujících částí kapitoly je zejména znalost způsobu, jakým aplikace uchovává načtené položky.

Každá načtená položka, resp. jen její údaje, které ji z pohledu aplikace dostatečně popisují, je v aplikaci uložena do patřičné datové struktury („*Folder*“ nebo „*File*“). Tyto struktury jsou seskupovány do vektoru těchto struktur, přičemž každá struktura typu „*Folder*“ obsahuje (mimo jiné) další vektor struktur typu „*File*“. Každý vektor v aplikaci je teoreticky limitován pouze rozsahem hodnot proměnné typu Integer.

7.2 Načtení položek

Načtení položek do nově vytvářeného archivu se provádí prostřednictvím nabídky v menu hlavního okna aplikace (bližší popis viz kapitola 6.1.1). Nyní bude vysvětlen postup, který zajišťuje správné uložení do struktur programu.

7.2.1 Načtení souborů

Je jednodušší verzí načítacího procesu. Uživatel prostřednictvím dialogového okna grafického uživatelského rozhraní vybere 0 – N souborů, kde N je libovolné celé číslo. Dialogové okno aplikaci vrátí seznam souborů, které uživatel vybral. Není-li seznam prázdný, následuje cyklus, při němž dochází k průchodu prvky tohoto seznamu. Pro každý prvek (1 prvek = 1 načtená položka) je vytvořena datová struktura „*Folder*“, která je následně naplněna podstatnými informacemi o tomto souboru. Naplněná struktura je potom přidána do „kořenového“ vektoru struktur „*Folder*“. Jsou-li takto načítány soubory, jsou vektory typu „*File*“, nacházející se v těchto strukturách, nevyužity.

7.2.2 Načtení složky

Je náročnější verzí načítacího procesu. Uživatel prostřednictvím dialogového okna grafického uživatelského rozhraní vybere 0 – 1 složku. Odkaz na ni je předán jako návratová hodnota tohoto dialogového okna. Tato hodnota je přijata hlavním oknem aplikace a následně zpracována. Načtená složka nemusí obsahovat jen soubory, ale mohou v ní být obsaženy také vnořené složky, které mohou obsahovat další vnořené složky atd. Maximální úroveň zanoření je závislá na konkrétním operačním (resp. souborovém) systému. Přidaná složka proto musí být zpracována rekurzivně.

Pro přidanou složku je vytvořena a naplněna nová struktura „*Folder*“. Následně dojde k získání seznamu odkazů na položky, které se v přidané složce nacházejí. Potom je tento seznam procházen a mohou být provedeny následující operace:

- Je-li ze seznamu načten soubor, je pro něj vytvořena a naplněna nová datová struktura „*File*“, která je následně přidána do vektoru položek aktuální složky. Dochází k načtení další položky v seznamu.
- Je-li ze seznamu načtena složka, je pro ni, stejně jako u souboru, vytvořena a naplněna nová datová struktura „*File*“, která je následně také přidána do položek aktuální složky. V tomto případě však nedojde k načtení další položky seznamu, ale dochází k rekurzivnímu volání operace pro přidávání položek, které je jako parametr předána cesta k této vnořené složce. Je-li při průchodu seznamu vnořené složky opět načtena další vnořená složka, je záznam o ní opět zanesen do vektoru struktur „*File*“ a opět dochází k další rekurzi, a tím pádem i k dalšímu zanoření. Když dospěje řízení programu při rekurzi ke „konečné složce“ (složka, která neobsahuje již žádné další vnořené složky), dojde ke zpracování všech souborů v ní obsažených. Následuje vnoření do nadřazené složky, ve které se pokračuje ve zpracovávání dalších položek v ní obsažených. Operace končí po zpracování veškerého obsahu vybrané složky (včetně obsahu všech složek této složce podřízených). Veškeré načtené položky jsou uloženy do vektoru struktur „*File*“, který je součástí struktury typu „*Folder*“ reprezentující v aplikaci hlavní načtenou složku.

Při načítání nových položek je současně aktualizován také obsah seznamu načtených souborů nacházející se v hlavním okně aplikace. Současně dochází k postupné aktualizaci proměnné, která uchovává informaci o počtu načtených položek. Identifikační čísla

položek, načtených do hlavního (kořenového) vektoru struktur „*Folder*“, jsou vždy rovna nule.

7.3 Proces archivace

Bezprostředně po zahájení procesu komprese dochází k vytvoření archivu, tj. souboru uživatelem stanoveného názvu s příponou „*uba*“ a s povolenou možností modifikace (není „pouze pro čtení“, anglicky „*Read Only*“). Potom dochází k zápisu co nejmenšího množství informací, které však plně postačují k bezpečné a správné rekonstrukci archivovaných dat.

Nejprve dochází k zápisu obecných dat, která, jak již bylo uvedeno výše, nejsou podstatná pro správné zpracování archivu, ale spíše podporují jistá „vylepšení“ a uživatelskou přívětivost archivátoru. Prvními zapsanými údaji jsou informace o hesle. Při otevření archivu archivátorem totiž dochází ke spuštění kontroly zabezpečení heslem. Po zápisu informací spojených s heslem následuje zápis informací spojených s komentářem (poznámkou) k archivu. Informace obecného rázu, které se archivu týkají, jsou tak přehledně udržovány na jednom místě.

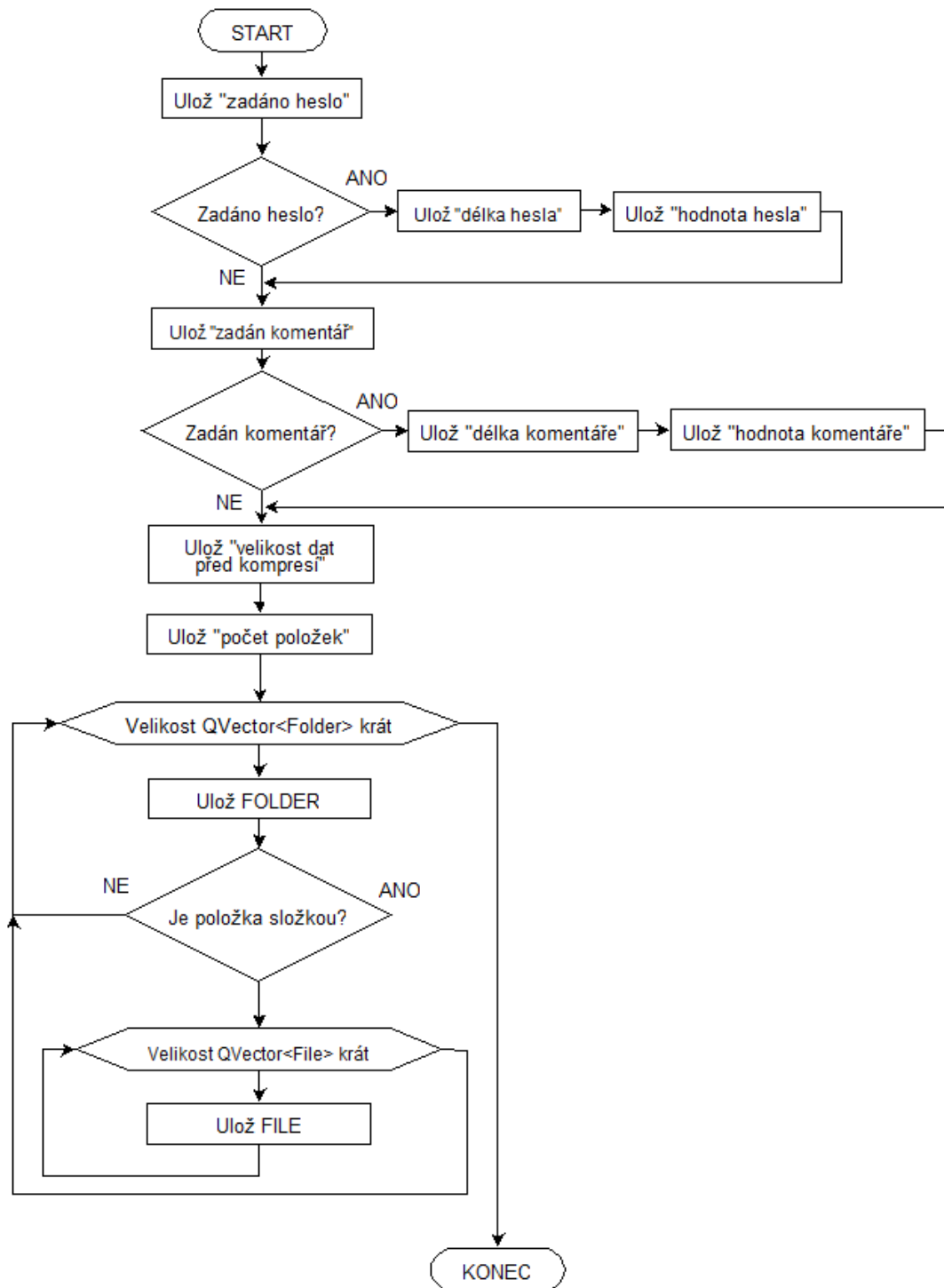
Bezprostředně poté následují údaje o původní velikosti a počtu položek, které se v archivu nacházejí. Údaj o počtu položek musí předcházet údajům o samotných položkách. Důvodem je využití tohoto záznamu v podmínce, která zajišťuje načtení položek, resp. informací, na základě kterých lze na disku z jednotlivých položek znovu zrekonstruovat původní soubory, jež tyto položky v aplikaci/archivu prezentují.

Následuje cyklus, při kterém dochází k postupnému průchodu vektoru struktur „*Folder*“ a vektoru struktur „*File*“, který je v každé této struktuře obsažený. Přitom dochází k postupnému zápisu údajů spojených s jednotlivými položkami. Tyto musí být zapsány dle platných pravidel aplikace, tzn. ve správném pořadí. V opačném případě by docházelo k nesprávnému zpracování takto vytvořeného archivu.

Výše uvedená funkčnost aplikace je popsána vývojovým diagramem. Ten je pro větší přehlednost je rozdělen na tři části – hlavní cyklus procesu archivace, způsob záznamu struktury „*Folder*“ a způsob záznamu struktury „*File*“.

7.3.1 Hlavní cyklus

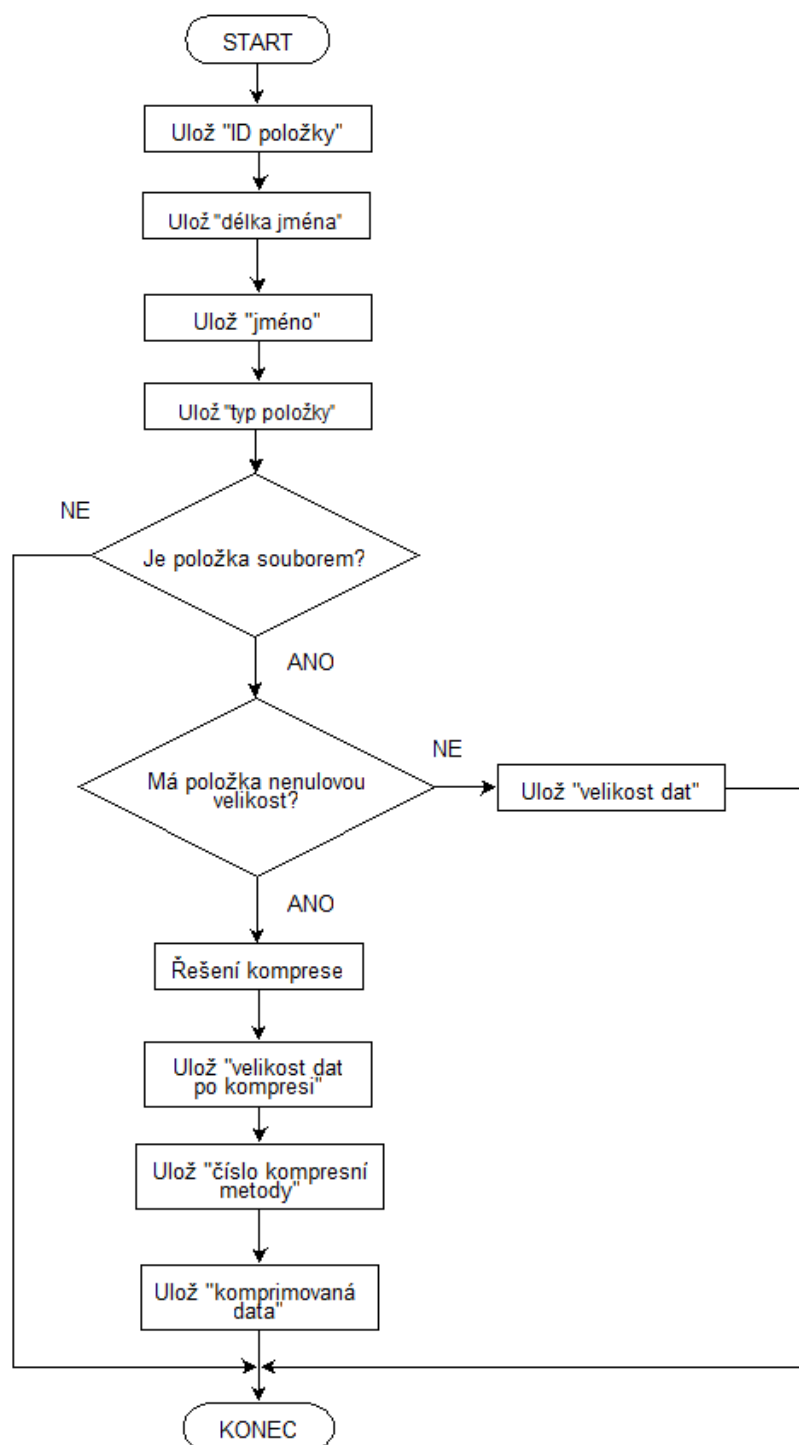
Zohledňuje způsob, resp. pořadí, zápisu jednotlivých informací do archivu. Vývojový diagram hlavního cyklu je znázorněn na Obr. 16.



Obr. 16. Vývojový diagram pro hlavní cyklus procesu archivace

7.3.2 Záznam struktury „Folder“

Blíže popisuje způsob zápisu informací spojených s položkou hlavního (kořenového) vektoru, tzn., že rozšiřuje krok „Ulož FOLDER“ hlavního cyklu. Vývojový diagram záznamu struktury „Folder“ je znázorněný na Obr. 17.



Obr. 17. Vývojový diagram pro záznam struktury „Folder“

7.3.3 Záznam struktury „File“

Blíže popisuje způsob zápisu informací spojených s položkou vektoru obsaženého v každém prvku hlavního (kořenového) vektoru, tzn., rozšiřuje krok „Ulož *FILE*“ hlavního cyklu.

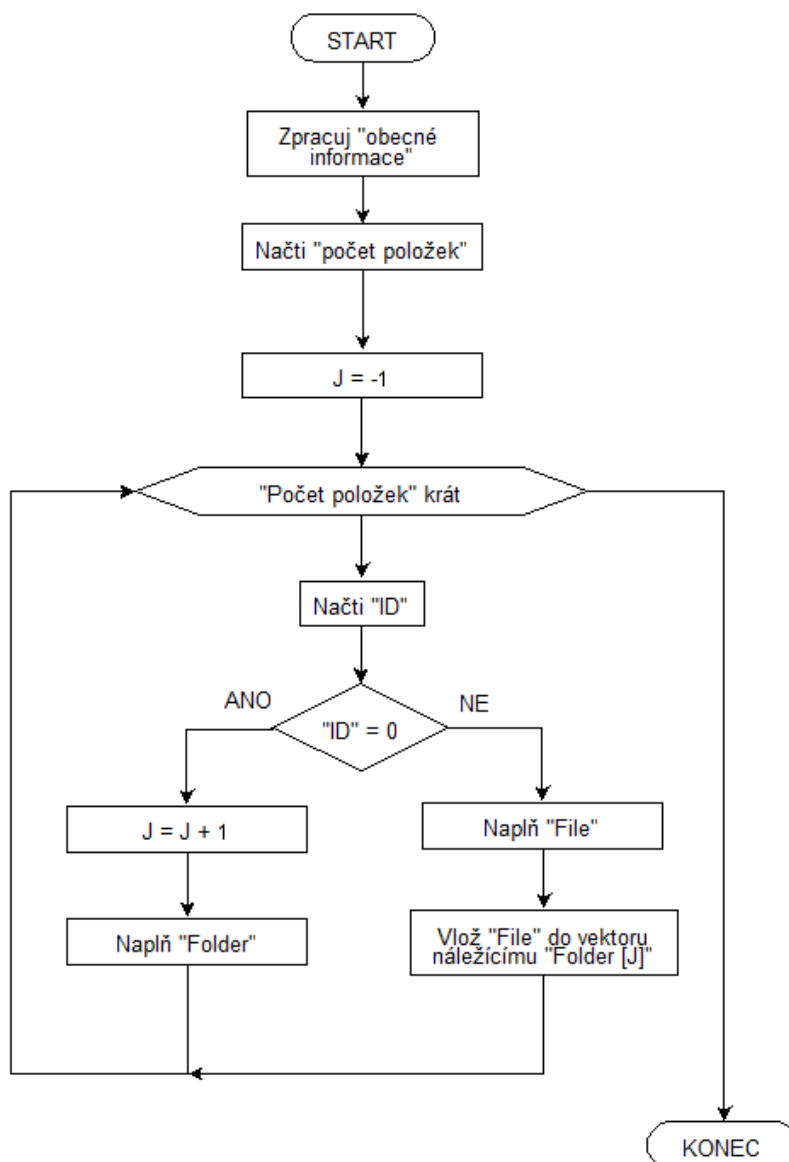
Zde není nutné uvádět celý vývojový diagram, neboť způsob záznamu struktury „File“ je téměř identický se způsobem záznamu struktury „Folder“. Jediným rozdílem je, že mezi prvními dvěma kroky (tzn. uložením ID položky a uložením délky jména položky) dochází k uložení dalšího údaje, tzv. „parent ID“, neboli identifikačního čísla nadřazené složky. Bližší popis významu a použití údaje „parent ID“ viz kapitola 8.3.

7.4 Načtení archivu

Stejně jako načítání položek do nově vytvářeného archivu, tak i načítání archivu se provádí prostřednictvím nabídky v menu hlavního okna aplikace (bližší popis viz kapitola 6.1.1). Záznamy jsou z archivu načítány ve stejném pořadí, v jakém byly do archivu při procesu archivace zaznamenávány. Jedinou výjimku tvoří obsahy archivovaných položek, které jsou načítány až v případě potřeby (při procesu extrakce). Bezprostředně po načtení archivu totiž tato data nemají žádný význam a pouze by zabírala paměť počítače. Jelikož je archiv binárním souborem, musí dojít k uložení pozice, na které začíná obsah konkrétního souboru v archivu. Využitím informace o pozici dat v souboru spolu s informací o velikosti těchto dat (resp. velikosti komprimovaného souboru) pak lze kdykoliv přistoupit k datům libovolného archivovaného souboru. V opačném případě by totiž muselo docházet k opětovnému průchodu obsahem archivu pro každou položku typu soubor, což by mohlo mít (zejména u archivů s velkým množstvím archivovaných položek) velmi nepříznivý dopad na rychlost procesu extrakce.

Při načítání položek v archivu se s výhodou využívá faktu, že kořenové položky, tzn. položky, které mají být uloženy v kořenovém vektoru, mají „ID“ rovno nule. Efektivně se tak rozlišuje, kdy končí jedna, resp. začíná další kořenová položka archivu.

Pro vysvětlení výše uvedeného principu plně postačuje zjednodušená (méně detailní) verze vývojového diagramu znázorněného na Obr. 18.



Obr. 18. Vývojový diagram pro načtení archivu

Poznámka k diagramu:

Proměnná „*J*“ je v diagramu uvedena jen z důvodu lepší názornosti. Jelikož jsou jednotlivé položky do archivu zapisovány systematicky (vždy položka z vektoru struktur „*Folder*“ následovaná položkami z vektoru struktur „*File*“ aktuální struktury „*Folder*“ náležící), jsou položky typu „*File*“ zapisovány vždy do posledního prvku vektoru struktur „*Folder*“.

7.5 Proces extrakce

Základními předpoklady pro správnou funkčnost tohoto procesu jsou:

- při načtení archivu (viz kapitola 7.4) došlo ke správnému a úplnému naplnění kořenového vektoru
- aplikace má stále k dispozici přístup k načtenému archivu

Při procesu extrakce dochází k postupnému průchodu kořenového vektoru, přičemž samotný postup extrakce závisí na konkrétním typu položky (složka nebo soubor). Vzhledem ke způsobu implementace je vhodné rozdělit popis postupu také podle vektoru, v němž se extrahovaná položka nachází:

7.5.1 Extrakce kořenové složky (struktura „Folder“)

V kořenovém adresáři, kam je archiv extrahován, je vytvořena složka s názvem, který je pro ni přidělen (uložen ve struktuře v proměnné „name“). Poté dochází k zavolání funkce sloužící pro zpracování položek vektoru struktur „File“, který náleží této struktuře. Jako parametr jí je, mimo jiné, předána také cesta k této složce. Ke zpracovávání další položky „Folder“ dochází až po kompletním zpracování všech položek „File“ v aktuální položce obsažených.

7.5.2 Extrakce kořenového souboru (struktura „Folder“)

Jak již bylo v předchozím textu uvedeno, při načítání archivu se neukládají obsahy souborů do paměti, ale ukládá se pouze odkaz na pozici těchto dat v archivu. Je-li aktuální položka souborem, je na základě tohoto údaje přenastavena pozice ukazatele v archivu. Následuje překopírování požadovaného obsahu archivovaného souboru do dočasného souboru „temp_in.tmp“ (blíže viz kapitola 7.6). Ten je poté dekomprimován a takto dekomprimovaný obsah je zapsán do souboru, který při extrakci vznikne v kořenové složce.

7.5.3 Extrakce vnořené složky (struktura „File“)

Je prakticky identickým postupem jako při extrakci kořenové složky. Vzhledem ke skutečnosti, že v rámci zpracování vnořených složek může docházet k dalším

zanořováním do předem neurčené hloubky, je funkce pro zpracování podřízených položek uzpůsobena pro rekurzivní volání. Funkce je tak vždy volána s parametry, mezi nimiž je také cesta ke složce, do níž mají být následně položky extrahovány.

Princip extrakce spočívá na schopnosti programu detekovat typ aktuálně zpracovávané položky načtené při průchodu prvků vektoru struktur „*File*“. Je-li detekována skutečnost, že aktuální zpracovávaná položka je složkou, dojde k rekurzivnímu volání výše zmiňované funkce pro zpracování podřízených položek. Proces extrakce končí po průchodu a extrakci všech prvků vektoru struktur „*Folder*“ a jim podřízených vektorů struktur „*File*“.

7.5.4 Extrakce vnořeného souboru (struktura „*File*“)

Taktéž se jedná o prakticky identický postup jako při extrakci kořenového souboru. Jediným rozdílem je pouze skutečnost, že vnořené soubory jsou, stejně jako vnořené složky, extrahovány prostřednictvím funkce pro zpracování podřízených položek. Nedochozí tak tedy k vytváření extrahovaných souborů v kořenové složce, ale tyto soubory jsou vytvářeny v patřičných složkách, kterým, dle původní stromové struktury, náleží.

Při extrakci položek z archivu musí být zajištěno, že tyto položky budou tvořit identickou stromovou strukturu jako jejich originální předloha. Toho je docíleno využitím dvojice informací náležící každému prvku, který reprezentuje složku nebo soubor, tzn. „*ID položky*“ a „*parent ID*“ (neboli identifikační číslo nadřazené složky). Jedinou výjimku tvoří struktury „*Folder*“ (kořenový vektor), jelikož se zde předpokládá, že neexistuje žádná složka, která by jim byla nadřazena. Bližší popis výše zmiňovaného způsobu implementace bude vysvětlen v kapitole 8.3.

7.6 Dočasné soubory

Aplikace při procesu archivace/extrakce s výhodou využívá dvou tzv. dočasných souborů (anglicky „*temporary files*“). Jedná se o soubory s názvy „*temp_in*¹⁶“ (používán jako

¹⁶ Složeno z „*temp*“, tj. zkratka slova „*temporary*“ (česky „*dočasný*“) a „*in*“ (česky „*v*“ nebo „*do*“).

vstupní soubor) a „*temp_out*“¹⁷ (používán jako výstupní soubor), které, jak již jejich pojmenování napovídá, se využívají pro krátkodobé uchování dat. Pro jejich snadnou odlišitelnost od ostatních souborů byla zvolena přípona „*tmp*“. Nejsou-li dočasné soubory ve složce aplikace k dispozici, jsou zde aplikací, v případě potřeby, automaticky vytvořeny. Po korektním ukončení procesu archivace/extrakce jsou dočasné soubory automaticky vymazány.

Při procesu archivace dochází k načítání dat přímo ze zdrojového souboru. Výsledná (komprimovaná) data jsou zapisována do dočasného souboru „*temp_out.tmp*“. Po dokončení procesu archivace je zjištěna velikost komprimovaných dat a tento údaj je společně s datovým obsahem dočasného souboru zapsán do archivu.

Dekomprese začíná, jak již zde bylo uvedeno, nalezením začátku obsahu patřičného souboru v archivu. Poté je tento obsah vykopírován do dočasného souboru „*temp_in.tmp*“, který je následně předán patřičné dekompresní metodě. Výstup metody je následně zapisován přímo do patřičného souboru, který má být dekompresí vytvořen.

Dočasných souborů (konkrétně souboru „*temp_out.tmp*“) využívá také implementovaný rozhodovací mechanismus, který bude blíže popsán v kapitole 8.2.

Výše zmiňovaný způsob má sice negativní vliv na rychlost procesu archivace/extrakce, na druhou stranu však šetří paměťové prostory počítače, což má pozitivní vliv na jeho výkon.

V případě, že je zvolen způsob komprese „*Bez komprese*“ (dále viz kapitola 8.1.1), dochází k „přímému“ kopírování obsahu souborů, tzn. nevyužívají se dočasné soubory.

¹⁷ Složeno z „*temp*“, tj. zkratka slova „*temporary*“ (česky „*dočasný*“) a „*out*“ (česky „*ven*“).

8 OSTATNÍ IMPLEMENTACE

8.1 Implementované metody

Do aplikace byly implementovány následující bezztrátové kompresní metody:

8.1.1 Bez komprese

V rámci implementace jednotlivých kompresních algoritmů byla ponechána možnost vytvoření archivu z nezkomprimovaných obsahů souborů. Této možnosti lze s výhodou využít zejména v případě, kdy žádná z implementovaných metod nedokáže efektivně zkomprimovat vstupní obsah souboru. Bude-li archiv vytvářen nad velkým množstvím malých souborů, může uživatel využitím této funkcionality uspořít značný datový prostor na disku (viz kapitola 3.2) a to i navzdory faktu, že nedojde ke kompresi obsahů těchto souborů. Tvorba takového archivu navíc probíhá v relativně krátkém časovém intervalu. Uživatel může této možnosti využít také pro rychlé a jednoduché zabezpečení archivovaného obsahu (viz kapitola 8.6).

8.1.2 Run-Length Encoding (RLE)

Jedná se o jeden z nejjednodušší bezztrátových kompresních algoritmů. Pracuje na bázi zhušťování opakujících se znaků, přicházejících na vstup algoritmu. Výstupem RLE je posloupnost dvojic složených z tzv. proudového čísla a proudové hodnoty. Proudové číslo reprezentuje počet opakování proudové hodnoty, která za tímto číslem následuje. Funkcionalitu této metody lze jednoduše demonstrovat na následujících příkladech:

Příklad 1:

Uvažujme vstupní posloupnost znaků ve tvaru „aaaabbbb“, tedy řadu čtyř znaků hodnoty „a“, za níž bezprostředně následuje řada čtyř znaků hodnoty „b“. Pak je výstupem kompresního algoritmu RLE řetězec složený ze dvou dvojic proudového znaku a proudové hodnoty, tedy „4a4b“. Délka vstupního řetězce je 8 znaků, výstupní řetězec obsahuje 4 znaky. Kompresní metodou RLE bylo dosaženo komprese 50%.

Příklad 2:

Uvažujme vstupní posloupnost znaků ve tvaru „*pokus*“. Pak je výstupem kompresního algoritmu RLE řetězec složený z pěti dvojic proudového znaku a proudové hodnoty, tedy „*lp1o1kluls*“. Délka vstupního řetězce je 5 znaků, výstupní řetězec obsahuje 10 znaků. Kompresní metodou RLE bylo dosaženo dvojnásobné velikosti výstupu oproti vstupu.

Z výše uvedených příkladů je patrné, že s rostoucím množstvím shodných znaků, které bezprostředně za sebou následují, roste také efektivita RLE. Naopak při velmi častých změnách vstupních znaků efektivita RLE klesá. Lze tak konstatovat, že výstupem je vždy řetězec, jehož délka leží v intervalu $<2; 2 \cdot N>$, kde N je délka vstupního řetězce.

8.1.3 Adaptivní Huffmanovo kódování

Huffmanovo kódování je založeno na přidělování unikátních bitových kódů jednotlivým znakům v závislosti na četnosti jejich výskytu ve vstupním řetězci, přičemž nejkratší bitový kód mají ty znaky, které se ve vstupním řetězci vyskytují nejčastěji.

Do aplikace byla implementována adaptivní verze tohoto algoritmu. To znamená, že algoritmus nevyužívá předpokládanou pravděpodobnost, ale Huffmanův strom, který je vytvářen při průchodu vstupním řetězcem a na jehož základě pak dochází k přidělování unikátních kódů.

8.1.4 Lempel-Ziv-Welch (LZW)

„Komprimační algoritmus Lempel-Ziv-Welch (LZW) je jednou z nejrozšířenějších komprimačních metod, kterou používají (v různé formě) jak kompresní programy (např. ARJ, PKZIP, ZOO, LHA atd.), tak i různé grafické formáty obrázků. Jedná se o tzv. substituční (adaptivní slovníkovou) metodu.“[1]

LZW pracuje na bázi tvorby slovníku (nebo také kódovací tabulky) opakujících se slov. Během zpracovávání vstupních dat je, při prvním výskytu konkrétního řetězce znaků, tento načtený řetězec poslán na výstup v nezměněné formě. Při dalších výskytech identického řetězce je však na výstup odeslána pouze tzv. zástupová hodnota, která prezentuje tento identický řetězec.

8.2 Rozhodovací mechanismus

Jedním z požadavků na vytvářenou aplikaci byla také implementace mechanismu, který bude při kompresi volit nejvhodnější kompresní algoritmus speciálně pro každý komprimovaný soubor. Je-li tento mechanismus aktivován (je uživatelsky nastavena volba kompresního algoritmu na základě konkrétního parametru v dialogovém okně pro nastavení parametrů archivace), je spouštěn bezprostředně před samotnou „finální“ kompresí souboru, tzn. před kompresí celého souboru zvolenou metodou a následným zápisem tohoto souboru do archivu. Rozhodovací mechanismus pracuje na základě porovnávání výsledků kompresního procesu vzorku souboru jednotlivých implementovaných kompresních algoritmů. Je vhodné upozornit na skutečnost, že rozhodovacím mechanismem je volen pouze kompresní algoritmus, nikoliv dekompresní. Ten je pevně dán zvolenou kompresní metodou.

8.2.1 Vzorek souboru

Vzorkem je část obsahu souboru, která tento soubor v rozhodovacím mechanismu reprezentuje. Minimální velikost vzorku není omezena, maximální velikost byla implicitně stanovena na 512 kB¹⁸. Hodnota byla zvolena s přihlédnutím k následujícím skutečnostem:

- s klesající velikostí vzorku klesá efektivita volby vhodné kompresní metody
- s rostoucí velikostí vzorku, tzn. komprimovaných dat, roste časová náročnost komprese
- hodnota není pevně dána, lze ji změnit prostřednictvím grafického uživatelského rozhraní aplikace (viz kapitola 6.1.2)

Získávání vzorku souboru se řídí předpokladem, že celý obsah souboru nemusí mít identickou strukturu, ale může například začínat tzv. hlavičkou¹⁹, která se může lišit od ostatního obsahu. Proto se vzorek získává z dat nacházejících se uprostřed souboru, kde lze s největší pravděpodobností očekávat výskyt obsahu, který soubor nejlépe

¹⁸ 1 kB = 1024 B, bliže viz kapitola 5.1.

¹⁹ Údaje spíše informativního charakteru.

prezentuje. Pozici v souboru, od níž dochází k načítání vzorku, lze stanovit matematicky jako $A = \frac{B}{2} - \frac{C}{2} = \frac{B - C}{2}$; kde A je výsledná pozice v souboru, B velikost vzorkovaného souboru a C maximální velikost vzorku.

V případě, že má soubor menší velikost, než je stanovená velikost vzorku, je rozhodovacímu mechanismu k testování předán celý jeho obsah.

8.2.2 Nastavení velikosti vzorku

Pro uchování uživatelsky modifikované hodnoty velikosti vzorku je použit binární konfigurační soubor „*setting.cfg*“ uložený v hlavní složce aplikace. Při spuštění aplikace dojde k ověření, zda tento soubor existuje. Pokud ano, je z něj načtena hodnota velikosti vzorku. V opačném případě je použita předem nastavená hodnota 512 kB. Hodnotu lze změnit prostřednictvím formuláře pro nastavení aplikace. Nová hodnota je ukládána do výše zmiňovaného konfiguračního souboru, který je, v případě potřeby, pro uchování této hodnoty nově vytvořen.

8.2.3 Princip mechanismu

Rozhodovací mechanismus ke svému chodu využívá vektor struktur typu „*MethodChoose*“ (viz kapitola 6.2.3). Je založen na principu postupné komprese konkrétního vzorku jednotlivými metodami implementovanými v aplikaci. Po každé kompresi konkrétní metodou dochází ke zjištění (příp. výpočtu) hodnot podstatných při volbě vhodné kompresní metody pro daný soubor, tj. kompresní poměr mezi vstupním a výstupním obsahem, doba trvání komprese a efektivita komprese za jednotku času²⁰. Tyto hodnoty jsou vloženy do datové struktury „*MethodChoose*“, která je následně vložena do vektoru těchto struktur. Implementované metody jsou volány v pevném pořadí, a proto pozice výsledků uložených ve vektoru korespondují s identifikačními čísly konkrétních metod, jejichž výsledky jsou na této pozici uloženy. Znamená to, že výsledky metody s identifikačním číslem rovným nule se nacházejí ve vektoru na první pozici (z pohledu vektoru na pozici 0), výsledky metody s identifikačním číslem 2 se ve vektoru nacházejí

²⁰ tyto parametry již byly vysvětleny při detailním popisu struktury „*MethodChoose*“.

na druhé pozici (z pohledu vektoru na pozici 1) atd. Po průchodu vzorkem všemi implementovanými metodami a následném uložení výsledků dochází k volbě nejvhodnější kompresní metody.

Podmínka, na jejímž základě dochází k volbě nejvhodnější metody, je závislá na volbě uživatele. Ten může, prostřednictvím uživatelského rozhraní, volit mezi „nejlepší“, „nejefektivnější“ nebo „nejrychlejší“ (význam možností viz kapitola 6.1.3) metodou pro konkrétní archivovanou položku (v tomto případě soubor s nenulovou velikostí). Po zvolení patřičné metody dochází ke kompresi celého obsahu daného souboru. Bezprostředně poté jsou zapsány patřičné údaje, tj. velikosti obsahu po kompresi, čísla zvolené metody a samotného obsahu, do výsledného archivu.

Jednotlivé archivované položky jsou zpracovávány postupně, přičemž pro každou takovou položku dochází k průchodu celým rozhodovacím mechanismem. Znamená to, že každý archivovaný soubor může být komprimován jinou kompresní metodou.

Je-li uživatelem zvolena konkrétní metoda, je celý výše popsany mechanismus ignorován a při archivačním procesu dochází přímo ke kompresi celého souboru konkrétní zvolenou kompresní metodou.

8.2.4 Optimalizace

Jak bylo uvedeno v předchozí části, je-li soubor menší velikosti, než je maximální velikost vzorku, jsou jednotlivé kompresní algoritmy testovány na celém obsahu tohoto souboru. Výstupem každého provedeného testu je tak výsledná podoba obsahu souboru po kompresi.

V rámci optimalizace tedy došlo k zavedení dvou proměnných do datové struktury „*MethodChoose*“. Jedná se o řetězcovou proměnnou „*fileContentAfter*“, kam je uložený výsledný řetězec testu konkrétní metody. Druhou proměnnou je logická proměnná „*wholeFile*“, která informuje o skutečnosti, že obsahem dříve zmiňované proměnné je komprimovaný obsah celého souboru. Je-li obsah souboru menší velikosti než je maximální velikost vzorku, je hodnota proměnné „*wholeFile*“ nastavena na hodnotu „*TRUE*“ (neboli „*pravda*“) a výsledek kompresního procesu je překopírován do proměnné „*fileContentAfter*“. V opačném případě nese proměnná „*wholeFile*“ hodnotu „*FALSE*“ (neboli „*nepravda*“) a proměnná „*fileContentAfter*“ zůstane prázdná.

Při vyhodnocování dosažených výsledků dojde k výběru nejvhodnější²¹ metody. Následuje zjištění, zda-li byl při testovacím kompresním procesu komprimován celý soubor (ověření hodnoty proměnné „*wholeFile*“ patřičné položky vektoru struktur „*MethodChoose*“). Je-li hodnota této proměnné pozitivní (tzn. „*TRUE*“), není již spuštěna další komprese, ale do výsledného archivu jsou jen překopírovány informace z patřičné položky výše zmiňovaného vektoru.

Tímto způsobem optimalizace došlo k jisté časové úspoře při komprimaci malých souborů. Tato skutečnost je nejvíce viditelná v případech, kdy je proces archivace prováděn nad velkým množstvím malých souborů. Podmínkou však je uživatelem akceptované přenechání volby kompresní metody rozhodovacímu mechanismu aplikace.

8.3 Stromová struktura

Jak již bylo řečeno, jedním z požadavků na aplikace zabývající se kompresí dat je úspěšná rekonstrukce komprimovaných dat. To znamená, že procesem extrakce, provedeném nad konkrétním archivem, musí dojít k naprosto identické rekonstrukci dat, která byla procesem archivace do tohoto archivu vložena. Z toho vyplývá, že v rámci složek musí také docházet k ukládání informací, na základě nichž lze při extrakci zrekonstruovat také stromovou strukturu, kterou tvoří soubory a složky v této složce obsažené. V této části budou blíže vysvětleny proměnné „*ID*“ a „*parent ID*“ datových struktur „*Folder*“²² a „*File*“, které umožňují uchovávat a posléze zrekonstruovat stromovou strukturu jednotlivých položek.

8.3.1 Shrnutí pojmů

Na úvod kapitoly je vhodné shrnout vysvětlení jednotlivých pojmů, které budou v následující části často používány:

²¹ Vztaheno k požadavku uživatele.

²² U datové struktury „*Folder*“ pouze proměnná „*ID*“.

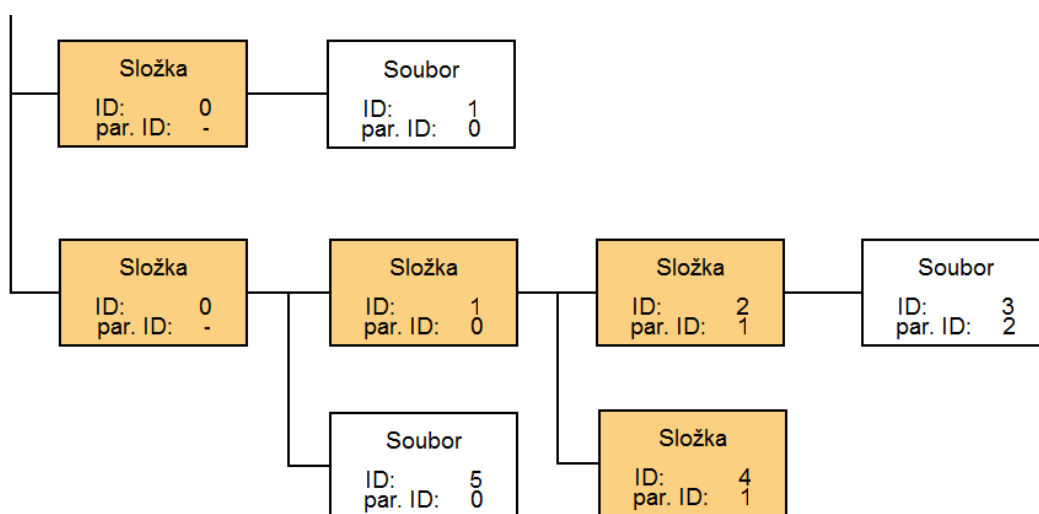
- **Kořenový (hlavní) vektor** – vektor struktur „*Folder*“, který je v aplikaci využíván. Jak již název „kořenový“ napovídá, lze průchodem tohoto vektoru přistoupit ke všem položkám, které byly do archivu načteny.
- **„*Folder*“** – datová struktura reprezentující tzv. kořenové položky, které jsou v archivu načteny. To znamená, že neexistuje žádná položka, která by byla této položce nadřazena. Z výše uvedeného důvodu neobsahuje struktura typu „*Folder*“ proměnnou „*parent ID*“. Představuje-li daná struktura neprázdnou složku, je seznam souborů a složek, které jsou v ní obsaženy, uložen ve vektoru struktur „*File*“, který každé této struktuře náleží. Jelikož jsou hodnoty proměnné „*ID*“ unikátní jen z pohledu vektorů struktur „*File*“, je proměnná „*ID*“ struktury „*Folder*“ vždy rovna nule.
- **„*File*“** – datová struktura reprezentující položky vnořené v kořenové položce typu složka. Každá struktura „*Folder*“ obsahuje právě jeden vektor struktur „*File*“, přičemž tento vektor již neobsahuje žádné další vektory. Znamená to, že v rámci jednoho vektoru „*File*“ jsou uchovány veškeré informace o stromové struktuře položky „*Folder*“.
- **ID** – identifikační číslo konkrétní položky. Hodnota „*ID*“ je inkrementována tak, aby žádné dvě různé položky ve vektoru struktur „*File*“ konkrétní struktury „*Folder*“ neměly hodnotu této proměnné identickou. Z pohledu přidělování „*ID*“ se nerozlišuje, jestli je tímto způsobem označována složka nebo soubor.
- **Parent ID** – hodnota proměnné je stanovena na základě identifikačního čísla složky, která je aktuální položce přímo nadřazena (ve které se přímo nachází).

8.3.2 Uchování

Načtením složky do aplikace dojde k vytvoření struktury „*Folder*“, která je naplněna podstatnými údaji o této složce. Následuje zavolání funkce pro zpracování podřízených položek, která zajistí rekurzivní průchod stromové struktury této složky. Všem položkám,

které se nacházejí přímo²³ v procházené složce, je jako „parent ID“ zadáno „ID“ rovno nule²⁴. To znamená, že se tyto položky nacházejí přímo ve složce, kterou reprezentuje struktura „Folder“. Je-li při aktuálním průchodu načtena položka typu složka, dochází k rekurzivnímu volání funkce pro zpracování podřízených položek. Takto načteným položkám bude přitom jako „parent ID“ nastaveno „ID“, které bylo před rekurzivním voláním výše zmiňované funkce přiděleno složce, ve které se nachází.

Problematiku přidělování „ID“ a „parent ID“ jednotlivým položkám lze jednoduše vysvětlit také graficky (viz Obr. 19).



Obr. 19. Grafické vyjádření problematiky přidělování „ID“ a „parent ID“

8.3.3 Rekonstrukce

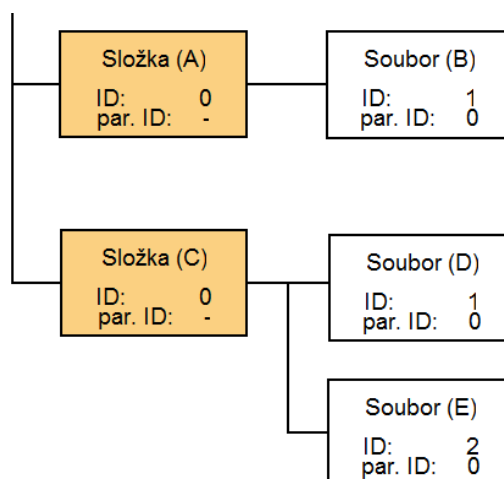
Pomocí proměnných „ID“ a „parent ID“ dochází také ke zrekonstruování kořenového vektoru struktur „Folder“ a v nich obsažených vektorů struktur „File“ při načítání archivu. Pro správné pochopení postupů rekonstrukce závislostí načtených položek je vhodné nejdříve zopakovat postup, jakým jsou jednotlivé položky vkládány do archivu:

Jak již bylo řečeno, při archivaci dochází k postupnému zpracovávání položek kořenového vektoru. Znamená to, že je do archivu nejdříve vložena kořenová položka

²³ Nejsou zanořené v další složce, která je podsložkou právě procházené složky.

²⁴ Prvek „Folder“ má vždy ID rovno nule.

a následně jsou postupně zpracovány a do archivu vloženy všechny položky v této položce obsažené (vektor struktur „File“). Až poté začne být zpracovávána další položka kořenového vektoru. Pro příklad lze uvést stromovou strukturu graficky znázorněnou na Obr. 20.



Obr. 20. Grafické vyjádření rekonstrukce struktury

Vzhledem k pravidlům, na jejichž základě probíhá proces archivace, by byly jednotlivé položky této stromové struktury zapsány do archivu v následujícím pořadí: A B C D E.

Proces načtení archivu je tak spjat s rekonstrukcí kořenového vektoru (včetně vektorů, které jednotlivé prvky tohoto vektoru obsahují), na základě kterého došlo k zápisu jednotlivých položek do archivu dle daného pořadí (ve výše uvedeném případě to znamená v pořadí A B C D E). Při rekonstrukci kořenového vektoru se s výhodou využívá informací o „ID“ a „parent ID“ a zejména pak skutečnosti, že:

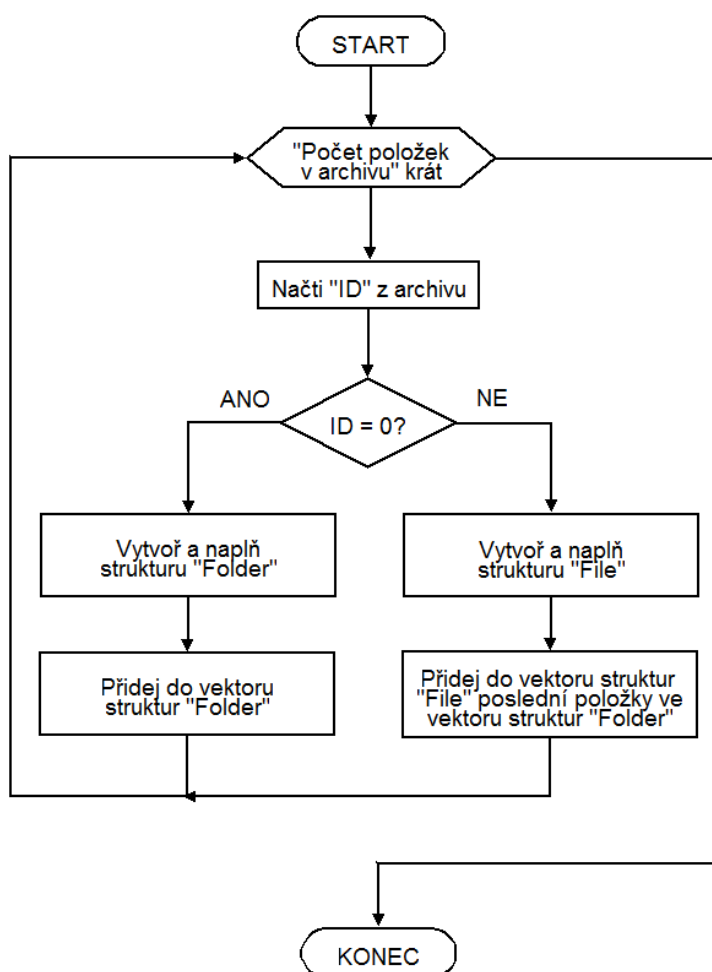
- položky kořenového vektoru mají hodnotu „ID“ rovnu nule.
- záznam každé položky uložené v archivu začíná informací o „ID“ této položky

Po zpracování obecných dat (mimo jiné je načten také údaj o počtu položek v archivu) dochází k načítání jednotlivých položek. Jak je v postupu načítání archivu uvedeno, prvním načteným údajem, který se týká konkrétní položky, je její „ID“. Je-li hodnota identifikačního čísla rovna nule, dojde k vytvoření struktury „Folder“. Ta je naplněna patřičnými údaji získanými z archivu a poté je vložena do kořenového vektoru. Tím je ukončeno zpracování jedné archivované položky a dochází k načtení další položky

z archivu. Opět je prvním načteným údajem „ID“ této položky. Zde mohou nastat dvě situace:

- Načtené identifikační číslo je rovno nule, což znamená, že se jedná o kořenovou položku. Je proto vytvořena nová struktura „Folder“. Ta je naplněna patřičnými informacemi a přidána do kořenového vektoru.
- Načtené identifikační číslo je rovno libovolnému přirozenému číslu různému od nuly, což znamená, že má být načtena položka podřízená položce kořenové. Proto je vytvořena nová struktura „File“, která je naplněna požadovanými údaji, a potom je vložena do vektoru struktur „File“ aktuální (poslední přidané) položky kořenového vektoru.

Výše popsanou funkcionalitu lze popsat vývojovým diagramem znázorněným na Obr. 21.



Obr. 21. Vývojový diagram rekonstrukce stromové struktury

8.3.4 Zobrazení položek v náhledu aplikace

Bezprostředně po načtení požadovaného archivu a naplnění patřičných vektorů aplikace dochází k zobrazení stromové struktury položek v komponentě typu `QTreeView`²⁵ nacházející se v levé části hlavního okna aplikace. Stromová struktura vytvořená v tomto pohledu je složena z jednotlivých objektů, přičemž jedna položka v pohledu je rovna jednomu objektu třídy `QStandardItem`²⁶. Tyto objekty mohou mezi sebou tvořit stromové závislosti, čehož se s výhodou využívá pro grafické znázornění stromové struktury načtených položek.

Daný způsob využívá principu rekurze. Příslušné funkci se předávají následující parametry:

- Kořenová položka, jejíž vektor struktur „*File*“ je zpracováván.
- „*ID*“ aktuální položky, které bude, po rekurzivním volání této funkce, figurovat jako „*parent ID*“.
- Objekt třídy `QStandardItem`, do kterého mají být přidávány jednotlivé položky.

Proces začíná vytvořením kořenové položky typu `QStandardItem`. Pokud je kořenová položka složkou, dochází k volání funkce „pro vytvoření stromové struktury v náhledu aplikace“. Zde dochází k průchodu vektoru struktur „*File*“ náležícímu aktuálně zpracovávané kořenové položce. Pro každý prvek tohoto vektoru dochází k ověření, zda-li je jeho „*parent ID*“ shodné s „*ID*“ složky, do které má být v pohledu aplikace přiřazen. K přiřazení dochází pouze v případě, jsou-li hodnoty těchto údajů shodné, přičemž průběh algoritmu, který jednotlivé položky zpracovává, je závislý na typu právě načtené položky takto:

²⁵ „Q“ v názvu značí, že jde o komponentu knihovny Qt; slova „Tree“ a „View“ po překladu do češtiny znamenají „strom“ a „pohled“. Lze tedy říct, že se jedná o komponentu sloužící pro zobrazení stromové struktury.

²⁶ Třída knihovny Qt.

- Položka reprezentující soubor je přiřazena objektu `QStandardItem`, který byl funkcí pro vytvoření stromové struktury v náhledu aplikace předán jako jeden z parametrů. Poté je načtena další položka...
- V případě, že položka reprezentuje složku, je postup zpracování shodný až do chvíle, kdy je tato položka reprezentovaná jako objekt třídy `QStandardItem` přiřazena nadřazenému objektu třídy `QStandardItem`. Před načtením další položky zde totiž nastává rekurzivní volání aktuálně popisované funkce, přičemž se jako parametry této funkce předává:
 - informace o zpracovávaném kořenovém prvku
 - ID přiřazené této položce
 - objekt třídy `QStandardItem`, který tuto položku reprezentuje

Z toho vyplývá, že při rekurzivním průchodu, který následuje, dochází k rozšíření této nově přidané položky reprezentující složku.

Výše popsaného mechanismu se využívá ve všech případech, kdy dochází k aktualizaci pole `QTreeView` zobrazujícího stromovou strukturu souborů načtených do aplikace, tzn. při přidávání souborů a složek a při načítání již vytvořeného archivu.

8.4 Vlákna

Při práci s aplikací dochází k časově náročným operacím, při kterých se musí současně aktualizovat údaje grafického uživatelského rozhraní, které uživatele pravidelně informují o průběhu těchto operací. V některých případech také musí být umožněno probíhající operaci předčasně ukončit. Z těchto důvodů byla do aplikace implementována vlákna (anglicky „*threads*“). Při časově náročných operacích (archivace, extrakce a načítání archivu/položek/složek²⁷) je spuštěno nové vlákno, které je určeno pouze k obsluze dané operace. Tím pádem může hlavní vlákno nadále obsluhovat jen uživatelské rozhraní.

²⁷ Projeví se zejména při velkém množství položek.

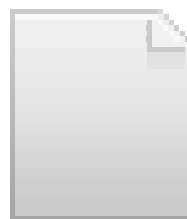
Vlákno, které zpracovává časově náročnou operaci, komunikuje s hlavním vláknem pomocí signálů. Hlavní vlákno tyto signály přijímá, přičemž každému tomuto signálu je přiřazena akce, která bude provedena (nejvýznamnější je aktualizace průběhu operace). Tímto způsobem implementace je uživateli umožněno sledovat průběh prováděné operace, příp. pracovat s uživatelským rozhraním (měnit nastavení zaškrtnutých políček zobrazeného dialogu²⁸ nebo předčasně ukončit operaci).

8.5 Rozlišení složek a souborů v pohledu

Jak již bylo uvedeno v popisu implementovaného uživatelského rozhraní, v hlavním okně aplikace se nachází tzv. pohled²⁹, v němž je zobrazena stromová struktura položek načtených v aplikaci. Pro větší přehlednost a informační přínosnost tohoto pohledu došlo ke grafickému odlišení typů položek v tomto pohledu zobrazených. Před každým názvem zobrazené položky je zobrazena patřičná ikona podávající informaci o tom, zda se jedná o složku (obr. 22) nebo o soubor (obr. 23). Volba typu ikony je závislá na logické proměnné „*isDirectory*“ nacházející se v obou strukturách využívaných pro reprezentaci položek v aplikaci, tj. „*Folder*“ a „*File*“.



Obr. 23. Ikona
složky



Obr. 22. Ikona
souboru

²⁸ Bližší informace o tomto dialogu viz kapitola 6.1.

²⁹ Komponenta třídy QTreeView.

8.6 Zabezpečení archivu

Archiv, vytvořený prostřednictvím popisované aplikace, lze zabezpečit heslem. Tato možnost je uživateli nabídnuta při nastavování parametrů procesu archivace. Údaje o hesle jsou prvními záznamy, které jsou do archivu uloženy, a to z následujícího důvodu:

Při načtení vytvořeného archivu do aplikace dojde k načtení prvního údaje. Tím je, jak již bylo uvedeno při popisu postupu a principů archivace, logická hodnota, která uvádí, zda je archiv zabezpečen heslem. Jestliže bylo tohoto bezpečnostního mechanismu při tvorbě archivu využito, je načtená hodnota rovna „*TRUE*“. V takovém případě následuje informace o délce (počtu znaků) hesla a pak následuje samotná hodnota tohoto hesla. Po načtení hodnoty hesla je načítání záznamů z archivu pozastaveno a uživatel je vyzván, aby zadal správné heslo k archivu. Zde mohou nastat 3 situace:

- uživatel zadal správné heslo; aplikace dále pokračuje v načítání záznamů z archivu a to dle platných pravidel uvedených v kapitole 7.4.
- uživatel zadal nesprávné heslo; aplikace stále vyžaduje zadání korektního hesla, načítání záznamů z archivu je stále pozastaveno.
- uživatel zrušil dialogové okno požadující zadání správného hesla; načítání archivu je ukončeno, což znamená, že spojení s archivem je uzavřeno a aplikace se vrací do výchozího stavu³⁰.

Mimo možnosti zabezpečení archivu heslem již nebyl implementován žádný jiný bezpečnostní mechanismus.

8.7 Blokace neaktuálních funkcí

V rámci implementace každého projektu musí dojít k ošetření všech nestandardních stavů, do kterých se aplikace může vlivem uživatelských akcí dostat. Uživateli nesmí být např. umožněno spouštět funkci pro extrakci archivu v době, kdy dosud žádný archiv do aplikace načten nebyl. Je několik způsobů, kterými lze tyto nestandardní situace ošetřit:

³⁰ Veškeré záznamy aplikace uložené v paměti jsou z této paměti vymazány, aplikace tak vypadá, jako by byla nově spuštěna.

- Zobrazení varovné zprávy v situaci, kdy by se uživatel pokusil spustit operaci, která je v danou chvíli neaktuální. Tato operace by nebyla spuštěna.
- Zablkováním možnosti spuštění této operace formou blokace nabídky v menu, příp. blokace tlačítka atd. Ta se projevuje grafickým odlišením od ostatních komponent daného typu tzv. „zašednutím“³¹.
- Skrývání komponent spouštějících operace, které nejsou v danou chvíli aktuální.
- Změna akce náležící danému tlačítku v závislosti na stavu aplikace. Některé aplikace mohou zamezovat nežádoucím změnám tím, že rozlišují více aplikačních stavů – prohlížení a editace údajů. Mezi těmito stavy je pak umožněno přepínat například tlačítkem, které mění svoji funkci mezi „oprav“ a „ulož“ na základě stavu, ve kterém se aplikace právě nachází.
- Kombinací výše uvedených způsobů.

Při implementaci této aplikace byla zvolena právě systematická kombinace prvních dvou uvedených způsobů upozorňování uživatele. Znamená to, že veškeré neaktuální nabídky uživatelského rozhraní jsou „zašedlé“ a varovné zprávy se zobrazují jen v případě systémových chyb (např. dojde-li k chybě při vytváření archivu) nebo při absenci zadaného údaje (např. nezadá-li uživatel, při nastavování parametrů procesu archivace, název archivu).

³¹ Vizualní vzhled komponenty je pozměněn do odstínů šedi a není umožněno ji používat, tzn. klikat na ni myší, apod.

9 ZHODNOCENÍ

Poslední kapitola této diplomové práce se zabývá popisem vlastností vytvořené aplikace, srovnáním výsledků kompresních procesů realizovaných touto aplikací s výsledky kompresních procesů vybraných archivačních programů a nástinem dalšího možného vývoje této aplikace.

9.1 Shrnutí výsledků implementace

Univerzální bezztrátový archivátor je aplikace závislá na operačním systému. Cílovým operačním systémem, určeným pro provoz této aplikace, je 32-bitová verze operačního systému Microsoft Windows. Na jiných operačních systémech (Linux, Unix, Mac OS) nebyla tato aplikace testována. Byla však navržena i s ohledem na možný budoucí požadavek snadné přenositelnosti na tyto operační systémy.

Do archivátoru byly implementovány 3 rozdílné bezztrátové kompresní algoritmy, ze kterých lze před zahájením samotného procesu archivace libovolně volit³². Volbu konkrétní kompresní metody lze také ponechat na implementovaném rozhodovacím mechanismu. Ten rozhoduje na základě uživatelem zvoleného parametru.

Archivátor umožňuje do vytvářeného archivu přidat vlastní komentář. Archiv je také možno zabezpečit prostřednictvím hesla. Obě tyto možnosti však musí být využity při tvorbě archivu, tzn., že musí být nastaveny v rámci dialogového okna pro nastavení parametrů archivace (viz kapitola 6.1.3). U již existujícího archivu však není možné dodatečně přidávat nebo modifikovat komentář ani dodatečně nastavovat nebo rušit zabezpečení heslem.

9.2 Srovnání aplikace s vybranými archivačními programy

Nejlepším způsobem, jakým lze zjistit kvality a nedostatky vytvořeného archivačního programu, je porovnání jeho výsledků s výsledky jiných archivačních programů.

³² Lze zvolit také archivaci „bez komprese“, tzn., že uživatel může volit jednu ze 4 nabízených možností.

9.2.1 Základní informace

Pro účely testování byly vybrány následující archivační programy: WinAce Archiver v2.69, WinBZip2 v1.0.0 (build 14), WinGZip v1.0.0 (build 21), WinRAR v3.93, WinUHA 2.0 RC1 (2005.02.27), WinZip 15.5 (9468).

V rámci srovnávání výsledků Univerzálního bezztrátového archivátoru s výše uvedenými archivátory bylo provedeno šest různých testů. V prvních čtyřech testech jsou porovnávány výsledky dosažené při kompresi konkrétního typu souboru. Pro tyto testy byly, z důvodu větší patrnosti rozdílů u jednotlivých výsledků, voleny soubory o velikostech v jednotkách MB. Pátý test je zaměřen na kompresi většího množství malých souborů. Poslední (šestý) test porovnává velikosti přídavných informací při archivaci jednoho souboru.

Dále byly, pro větší patrnost rozdílů ve výsledcích, jednotlivé testy prováděny na starším modelu notebooku s následující konfigurací³³:

- **Procesor:** Intel Celeron M 1,4 GHz
- **Operační paměť:** 512 MB
- **Pevný disk:** 5400 otáček/min., Ultra ATA
- **Operační systém:** Microsoft Windows XP SP3 (32-bit)

Univerzální bezztrátový archivátor, který vznikl v rámci této diplomové práce, bude v testech zastoupen dvakrát:

- **UBA** značí použití aplikace, kdy je vhodný kompresní algoritmus volen rozhodovacím mechanismem podle parametru „Kvalita“ (dále viz kapitola 6.1.3).
- **UBA*** značí použití aplikace s uživatelsky zvolenou kompresní metodou, přičemž tato metoda bude vybrána na základě výsledku rozhodovacího mechanismu pro daný typ souboru. Konkrétní zvolená metoda bude u každého testu uvedena.

³³ Ve výčtu parametrů notebooku jsou uvedeny pouze ty parametry, které mohou nejvíce ovlivnit dobu trvání kompresního procesu. Nejsou však uvedeny jednotlivé aplikace, které se spouští bezprostředně po spuštění operačního systému.

Pro zamezení ovlivňování výsledků kompresního procesu u jednotlivých výše uvedených archivačních programů bude archivace probíhat s implicitním nastavením těchto programů.

V průběhu testování byly sledovány a zaznamenávány 3 údaje:

- **Kompresní poměr** – poměr mezi velikostí vstupního a výstupního souboru vyjádřený v procentech
- **Čas komprese** – doba, za kterou dojde k úspěšnému provedení komprese
- **Čas dekomprese** – doba potřebná pro provedení dekomprese

9.2.2 Test 1 – spustitelný soubor

První test byl prováděn na spustitelném (anglicky „executable“) souboru počítačové hry. Velikost tohoto souboru byla 10 105 480 bajtů. Jednotlivé výsledky kompresních procesů různých archivačních programů jsou uvedeny v Tab. 6.

Program	WinAce	WinBZip2	WinGZip	WinRAR	WinUHA	WinZip	UBA	UBA*
Kompresní poměr (%)	83	86	86	83	83	85	98	
Čas komp. (mm:ss)	0:09	0:08	0:02	0:10	0:25	0:03	8:56	8:35
Čas dekomp. (mm:ss)	0:01	0:06	0:00	0:00	0:05	0:01	9:41	

Tab. 6. Výsledné hodnoty prvního testu

Pro tento soubor bylo rozhodovacím mechanismem zvoleno adaptivní Huffmanovo kódování.

9.2.3 Test 2 – 24-bitový BMP soubor

Druhý test byl prováděn na bitmapovém souboru s 24-bitovým rastrem o velikosti 3 932 214 B. Jednotlivé výsledky kompresních procesů různých archivačních programů jsou uvedeny v Tab. 7.

Program	WinAce	WinBZip2	WinGZip	WinRAR	WinUHA	WinZip	UBA	UBA*
Kompresní poměr (%)	3	3	5	3	3	5	10	
Čas komp. (mm:ss)	0:01	0:02	0:00	0:00	0:03	0:01	0:15	0:00
Čas dekomp. (mm:ss)	0:00	0:00	0:00	0:00	0:00	0:00	0:00	

Tab. 7. Výsledné hodnoty druhého testu

Pro tento soubor byl rozhodovacím mechanismem zvolen kompresní algoritmus LZW.

9.2.4 Test 3 – dokument DOC

Třetí test ověřuje schopnost archivačních programů komprimovat dokumenty, které mohou mimo standardní text obsahovat i různé jiné objekty (např. obrázky) nebo různě graficky modifikovaná písma (rozličné fonty, styly a barvy). Pro tento test byl zvolen dokument vytvořený v programu Microsoft Office Word. Dokument má příponu „doc“ a jeho velikost je rovna 4 467 200 bajtům. Výsledky testu viz Tab. 8.

Program	WinAce	WinBZip2	WinGZip	WinRAR	WinUHA	WinZip	UBA	UBA*
Kompresní poměr (%)	91	92	91	90	90	91	98	
Čas komp. (mm:ss)	0:04	0:04	0:01	0:03	0:11	0:02	4:12	3:47
Čas dekomp. (mm:ss)	0:00	0:03	0:00	0:00	0:02	0:01	4:04	

Tab. 8. Výsledné hodnoty třetího testu

Pro tento soubor bylo rozhodovacím mechanismem zvoleno adaptivní Huffmanovo kódování.

9.2.5 Test 4 – textový soubor

Bezprostředně po dokumentu byla otestována schopnost archivátorů zkomprimovat klasický textový soubor, jehož obsahem jsou čistě textové záznamy (přípona „.txt“). Velikost souboru byla 5 024 400 bajtů. Výsledky viz Tab. 9.

Program	WinAce	WinBZip2	WinGZip	WinRAR	WinUHA	WinZip	UBA	UBA*
Kompresní poměr (%)	4	3	5	3	3	4	24	
Čas komp. (mm:ss)	0:01	0:05	0:00	0:01	0:05	0:01	0:20	0:01
Čas dekomp. (mm:ss)	0:00	0:01	0:00	0:00	0:00	0:00	0:00	

Tab. 9. Výsledné hodnoty čtvrtého testu

Pro tento soubor byl rozhodovacím mechanismem zvolen kompresní algoritmus LZW.

9.2.6 Test 5 – 20 malých textových souborů

V předposledním testu byla odzkoušena schopnost archivátorů zkomprimovat větší množství malých textových souborů. Jedná se o 20 textových souborů, přičemž velikost každého z těchto souborů leží v intervalu $< 72; 52\,655 >$ bajtů. Celková velikost souborů je 214 371 bajtů a jejich obsah tvoří část nezkompilovaných zdrojových kódů Univerzálního bezztrátového archivátoru. Výsledky pátého testu se nacházejí v Tab. 10.

Program	WinAce	WinBZip2	WinGZip	WinRAR	WinUHA	WinZip	UBA	UBA*
Kompresní poměr (%)	16	X	X	22	15	22	57	59
Čas komp. (mm:ss)	0:00	X	X	0:00	0:00	0:00	0:09	0:00
Čas dekomp. (mm:ss)	0:00	X	X	0:00	0:00	0:00	0:00	0:00

Tab. 10. Výsledné hodnoty pátého testu

Vzhledem k většímu množství archivovaných souborů bylo při kompresním procesu použito více kompresních metod. Z tohoto důvodu se ve sloupci **UBA*** Tab. 10. nachází výsledky kompresního algoritmu, který dokázal soubory jako celek zkomprimovat nejlépe, tj. byl zvolen algoritmus LZW.

Programy WinBZip2 a WinGZip neumožňují vytvářet archiv z více souborů, a proto u těchto programů nejsou uvedeny výsledné hodnoty tohoto testu.

9.2.7 Test 6 – přídatné informace:

V rámci posledního testu (výsledky viz Tab. 11.) bylo zjišťováno, kolik přídatných dat je do archivu vloženo při archivaci textového souboru o velikosti 1 B – soubor s názvem „file.txt“, jehož obsahem je pouze 1 znak.

Program	WinAce	WinBZip2	WinGZip	WinRAR	WinUHA	WinZip	UBA
Velikost (B)	97	37	30	73	77	151	43

Tab. 11. Výsledné hodnoty šestého testu

9.3 Zhodnocení dosažených výsledků

U jednotlivých provedených testů se často objevuje časový údaj ve tvaru „0:00“. Tato nulová hodnota znamená, že konkrétní proces archivace/extrakce proběhl na testované sestavě v čase kratším než 1 vteřina.

Souhrnem výsledků jednotlivých testů lze jednoznačně dospět k následujícím závěrům:

- U všech testovacích dat bylo kompresí dosaženo jistého (byť někdy jen naprosto minimálního) pozitivního vlivu na jejich velikost. Znamená to, že velikost daná součtem vstupních dat po kompresi a tzv. přídatných dat je menší, než velikost vstupních dat před kompresí. Skutečnost, že u všech vzorků různých typů dat došlo při procesu archivace k pozitivnímu vlivu na velikost tohoto vzorku, neznamená, že archivátor dokáže efektivně zkomprimovat všechny soubory těchto testovaných typů.
- V testech, kde je pro kompresi zvoleno adaptivní Huffmanovo kódování, dochází k obrovským rozdílům mezi časy komprese/dekomprese Univerzálního bezztrátového archivátoru a ostatními archivačními programy.
- V žádném z uvedených testů nedosahuje Univerzální bezztrátový archivátor lepších výsledků, než konkurenční archivační programy. V testech číslo 1, 2 a 3 se však kvalita komprese provedené Univerzálním bezztrátovým archivátorem velmi přibližuje k výsledkům ostatních archivačních programů.

- Z pohledu výsledků jednotlivých testů je zřejmé, že Univerzální bezztrátový archivátor je nejefektivnější při kompresi bitmapových obrázků.
- Při archivaci jednoho souboru o velikosti 1 B je ukládáno, oproti ostatním testovaným archivačním programům, relativně málo přídavných informací. Méně informací do vytvářeného archivu přidávají jen archivační programy WinBZip2 a WinGZip. Naopak Univerzální bezztrátový archivátor ukládá v tomto případě 3,5 krát méně přídavných informací oproti programu WinZip.

9.4 Zhodnocení dalšího vývoje aplikace

V rámci písemné části diplomové práce byl popsán aktuální stav vytvořené aplikace, tzn. popis všech postupů a použitých datových struktur, veškeré výhody a omezení aplikace, význam a funkcionality celého uživatelského rozhraní, implementace jednotlivých operací a také implementované kompresní algoritmy. Na závěr byly provedeny jednotlivé testy, které umožnily porovnat vytvořenou aplikaci s jinými archivačními programy. Na základě všech těchto informací tak lze navrhnout možnosti dalšího vývoje.

Navzdory využití multiplatformní knihovny pro tvorbu grafického uživatelského rozhraní je vytvořená aplikace spustitelná pouze pod operačním systémem Microsoft Windows, což je silná konkurenční nevýhoda vůči multiplatformním aplikacím. Za jeden z dalších cílů vývoje lze proto považovat zajištění přenositelnosti aplikace mezi různými typy operačních systémů.

S přihlédnutím k výsledkům provedených testů lze adaptivní Huffmanovo kódování považovat za slabý článek kompresních metod. Přeprogramování této kompresní metody by tak mohlo zvýšit její efektivitu, zejména pak rychlost.

Za velmi významné vylepšení aplikace lze považovat přidání dalších kompresních algoritmů. S rostoucím počtem metod by sice rostla časová náročnost rozhodovacího mechanismu aplikace, na druhé straně by však při volbě nejvhodnější metody docházelo k výběru z většího množství různých metod. Tato skutečnost by měla pozitivní vliv na efektivitu volby nejvhodnější metody, a tím pádem by vedla ke zlepšení výsledků procesu archivace.

Položky, které jsou přidány do pohledu aplikace, již nejdou z tohoto pohledu odstranit³⁴. Umožněním odstranění konkrétní položky z pohledu hlavního okna aplikace by došlo ke zvýšení uživatelské přívětivosti této aplikace.

Dále lze například implementovat kontrolní ověřování archivu (tzv. „Checksum“, který by sloužil k ověření, zda-li není archiv poškozen), dle typu a pořadí v abecedě řadit jednotlivé přidávané položky v pohledu, umožnit dělení velkého archivu na větší množství dílčích souborů atd. Existuje velké množství způsobů, kterými lze docílit zvýšení efektivity nebo uživatelské přívětivosti aplikace. Na základě tohoto tvrzení však nelze mluvit o tom, že by aplikace „nebyla dokončena“. Tato skutečnost totiž není pouze výsadou popisované aplikace, ale platí pro každou aplikaci, která kdy byla, je nebo bude vytvořena, přičemž s rostoucí komplexitou konkrétní aplikace roste také množství potenciálních vylepšení a zdokonalení.

³⁴ Lze pouze vymazat celý pohled hlavního okna aplikace pomocí nabídky menu „Soubor → Nový archiv“. Tím však dojde k úplnému vymazání všech informací z aplikace.

ZÁVĚR

Cílem diplomové práce bylo vytvoření Univerzálního bezztrátového archivátoru. V něm mělo být implementováno několik rozdílných kompresních algoritmů a rozhodovací mechanismus, jehož úkolem je mezi těmito algoritmy volit ten nejvhodnější algoritmus pro konkrétní typ vstupních dat. Pro tyto účely byly do aplikace implementovány celkem tři různé algoritmy, které komprese dosahují odlišnými způsoby.

V závěru práce bylo provedeno testování vytvořené aplikace. Výsledky těchto testů byly poté porovnány s výsledky vybraných komerčních archivátorů. Při těchto testech bylo zjištěno, že vytvořený archivátor sice funguje korektně, ale na poli archivačních programů není příliš konkurenceschopný. To je však způsobeno zejména faktem, že Univerzální bezztrátový archivátor vznikl jako individuální diplomová práce po dobu několika měsíců. Porovnáván však byl s archivačními programy a jimi používanými kompresními mechanismy, které jsou vyvíjeny a zdokonalovány již několik (desítek) let – např. WinZip, se kterým byl Univerzální bezztrátový archivátor také porovnáván, vznikl již od roku 1991. Vzhledem k výše uvedeným výsledkům proto byly v rámci této diplomové práce diskutovány také možnosti dalšího vývoje aplikace, které by zlepšily zejména její konkurenceschopnost.

Srovnávacími testy archivátorů se prokázalo dosažení základních cílů, které byly stanoveny na začátku vývoje této aplikace. Při implementačním procesu bylo dbáno také na možnost případného rozšíření. Proto lze na dosažené výsledky mé práce plynule navázat a pokračovat v dalším vývoji a zdokonalování této aplikace.

Tvorba aplikace sloužící pro kompresi dat je velice složitou záležitostí, čemuž také odpovídá větší rozsah praktické části diplomové práce.

CONCLUSION

The objective of this thesis was to create a universal lossless archiver. In this archiver, there should be implemented the several different compression algorithms and decision-making mechanism, whose task is to choose between these algorithms the most appropriate algorithm for a particular type of input data. For this purposes, there were implemented three different algorithms which are achieving the compression by different ways.

At the end of this thesis, there were performed the comparative tests, which were headed to test the created application. The results of these tests were compared with results of another selected commercial archivers. During these tests it was found that the created archiver works correctly, but, on the field of archiving programs, there is not too much competitive. This is mainly due to the fact that the Universal lossless archiver was created as an individual thesis for several months. However, this archiver was compared with the archiving programs and their's compression mechanisms, which were developed and improved for several (dozen) years – for example, archiving program called like WinZip was created and improved since 1991. In the view of the above results in this thesis there were also discussed the possibilities of further development of this application, that would improve its competitiveness in particular.

Comparative tests showed accomplishment of the basic objectives, that were set at the beginning of developing the application. During the implementation process, the care has been taken also to the extension possibilities. Therefore, the results of my work may be followed up and anyone else may continue in the further development and improvement of this application.

Making an application, which is used to data compression, is a very complex matter, that is the reason of practical's part greater extent.

SEZNAM POUŽITÉ LITERATURY

- [1] MORKES, David . *Komprimační a archivační programy*. Vydání první. Brno: Computer Press, 1998. 177 s. ISBN 80-7226-089-8
- [2] HEROUT, Pavel . *Učebnice jazyka C*. Třetí upravené vydání. Příbram : KOPP, 1994. 269 s. ISBN 80-85828-21-9
- [3] GRIM, Ondřej . *Univerzální cizojazyčný slovník*. Bakalářská práce. Brno: FIT VUT v Brně, 2009
- [4] BLANCHETTE, Jasmin, SUMMERFIELD, Mark . *C++ GUI Programming with Qt 4*. Westford, Massachusetts: Trolltech ASA, 2008. 752 s. ISBN-13 978-0-13-235416-5
- [5] SUMMERFIELD, Mark . *Advanced Qt Programming: Creating Great Software with C++ and Qt 4*. Westford, Massachusetts: Addison-Wesley, 2010. 536 s. ISBN 978-0-321-63590-7
- [6] SALOMON, David, Giovanni, Motta . *Handbook of Data Compression*. Fifth Edition. London: Springer-Verlag London Limited, 2010. 1359 s. ISBN 978-1-84882-902-2

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

B Bajt

min. Minuta

kB Kilobajt

KiB Kibibajt

SEZNAM OBRÁZKŮ

Obr. 1. Originální obrázek v plné kvalitě	14
Obr. 2. Komprimovaný soubor, kvalita 50%	15
Obr. 3. Komprimovaný soubor, kvalita 1%	15
Obr. 4. Příklad vzhledu příkazové řádky	22
Obr. 5. Příklad vzhledu textového uživatelského rozhraní	23
Obr. 6. Příklad vzhledu grafického uživatelského rozhraní	24
Obr. 7. Popis hlavního okna aplikace	31
Obr. 8. Vzhled hlavního okna aplikace po načtení archivu	33
Obr. 9. Popis dialogu pro nastavení aplikace	34
Obr. 10. Popis dialogu pro nastavení parametrů archivace	35
Obr. 11. Popis dialogu pro nastavení poznámky	38
Obr. 12. Popis dialogu pro načítání zdrojových dat	39
Obr. 13. Popis argivačního/extračního dialogu	40
Obr. 14. Popis dialogu pro zadání hesla	41
Obr. 15. Popis dialogu s informacemi o aplikaci	41
Obr. 16. Vývojový diagram pro hlavní cyklus procesu archivace	52
Obr. 17. Vývojový diagram pro záznam struktury „Folder“	53
Obr. 18. Vývojový diagram pro načtení archivu	55
Obr. 19. Grafické vyjádření problematiky přidělování „ID“ a „parent ID“	66
Obr. 20. Grafické vyjádření rekonstrukce struktury	67
Obr. 21. Vývojový diagram rekonstrukce stromové struktury	68
Obr. 22. Ikona souboru	71
Obr. 23. Ikona složky	71

SEZNAM TABULEK

Tab. 1. Popis proměnných datové struktury „Folder“	42
Tab. 2. Popis proměnných datové struktury „File“	43
Tab. 3. Popis proměnných datové struktury „MethodChoose“	44
Tab. 4. Popis obecných dat	46
Tab. 5. Popis dat položek	47
Tab. 6. Výsledné hodnoty prvního testu	76
Tab. 7. Výsledné hodnoty druhého testu	77
Tab. 8. Výsledné hodnoty třetího testu	78
Tab. 9. Výsledné hodnoty čtvrtého testu	79
Tab. 10. Výsledné hodnoty pátého testu	80
Tab. 11. Výsledné hodnoty šestého testu	81

SEZNAM PŘÍLOH

Příloha 1. DVD se zdrojovými kódy a spustitelnou verzí aplikace.