

České vysoké učení technické v Praze
Fakulta elektrotechnická



Diplomová práce

Kompresní algoritmy, jejich implementace a vizualizace

Ondřej Hanousek

Vedoucí práce: Ing. Miroslav Balík, Ph.D.

Studijní program: Informatika a výpočetní technika

prosinec 2005

Poděkování

Na tomto místě bych rád vyjádřil své poděkování vedoucímu práce Ing. Miroslavu Balíkovi za čas, doporučení i rady, které mi poskytl a také bych rád poděkoval své rodině a blízkým za podporu během studia.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v příloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Dolním Bukovsku dne 2.1. 2006

.....

Abstract

The goal of this work is visualisation of lossless compression algorithms using Java applets and HTML pages. Second point is implementation of compression algorithms as Java streams.

Abstrakt

Práce se zabývá vizualizací bezztrátových kompresních algoritmů a možnostmi demonstrace kompresních algoritmů s využitím Java appletů a HTML stránek. Součástí jsou i implementace vybraných algoritmů pro univerzální použití v Javě.

Obsah

Seznam obrázků	xiii
1 Úvod	1
2 Cíle práce	2
3 Teoretický základ komprese dat	3
3.1 Shannonova teorie informace	3
3.2 Kódování celých čísel	5
3.3 Další pojmy	6
4 Návrh řešení	7
4.1 Použité technologie	7
4.2 Uživatelské rozhraní	7
4.3 Grafický výstup	9
4.4 Ovládání simulace	9
4.5 Implementace kompresního a dekompresního algoritmu v jazyce Java	10
5 Použité postupy	11
5.1 Webové stránky	11
5.2 Grafický výstup	11
5.3 Spouštění pomocí Java Web Start	12
5.4 Lokalizace	12
5.5 Krokování algoritmu	13
5.6 Statistický model dat	13
5.7 Implementace kompresních a dekompresních algoritmů	13
6 Statistické kompresní metody	14
6.1 Shannon-Fanovo kódování	15
6.2 Statické Huffmanovo kódování	16
6.3 Adaptivní Huffmanovo kódování	18
6.4 Aritmetické kódování	20
7 Slovníkové metody komprese dat	23
7.1 LZ 77	23
7.2 LZ 78	25
8 Závěr	27
9 Seznam literatury	29
A Seznam použitých zkratk	31
B Obsah příloženého CD	33
B.1 Text diplomové práce	33
B.2 Aplikace	33

Seznam obrázků

4.1	Základní uživatelské rozhraní	7
6.1	Komprese s využitím statistického modelu	14
6.2	Závěr Shannon-Fanova kódování	16
6.3	Závěr statického Huffmanova kódování	17
6.4	Závěr adaptivního Huffmanova kódování	20
6.5	Ukázka aritmetického kódování	21
7.1	Kroky algoritmu LZ 77	24
7.2	Slovník při kódování LZ 78	26

1 Úvod

Každým okamžikem vzniká na světě obrovské množství dat a každý ukládací prostor je jednou nedostatečný a každá kapacita linky je jednou plně vytížená. Možností, jak dosáhnout úspory datového toku nebo velikosti uložených dat je komprese dat.

Snahu o úsporné kódování najdeme už v Morseově abecedě z roku 1838, kde jsou kratší kódová slova přiřazena znakům, které se v angličtině nejčastěji vyskytují. O moderní kompresi dat však můžeme mluvit až od 40. let minulého století, kdy byla publikována informační teorie C. Shannona. Od té doby byla vyvinuta celá řada algoritmů bezztrátové komprese dat, které umožňují výrazné snížení velikosti datových souborů bez ztráty informace - v textových souborech vede využití komprese ke snížení velikosti přibližně na $1/3$ velikosti původní, pro obrazové informace ve formě fotografií přibližně na $1/2$ původní velikosti. Vzhledem k velkému rozšíření kompresních algoritmů by měla jejich znalost patřit ke znalostem studenta informatiky. Vhodnou formou vizualizace těchto algoritmů je možné dosáhnout lepšího vysvětlení funkce algoritmu než s využitím omezeného prostoru knihy nebo přednáškového slidu.

2 Cíle práce

Přesto, že existuje velké množství literatury ve formě knih i webových prezentací, které se zabývají kompresí dat a kompresními algoritmy, není v běžném formátu vždy dost dobře možné spojit ilustrující obrázek s popisem algoritmu a doprovodným textem. Cílem této práce je především návrh a implementace vhodného univerzálního způsobu pro vizualizaci algoritmu a s jeho použitím vizualizace vybraných kompresních algoritmů. Vizualizace je navíc vhodně doplněna souvisejícími teoretickými základy a souvislostmi. Výsledky vizualizace jsou navíc připraveny tak, aby bylo možné je použít v tištěné publikaci. Dalším cílem je pak implementace vybraných algoritmů v jazyce Java.

3 Teoretický základ komprese dat

3.1 Shannonova teorie informace

Od roku 1948 a publikace Shannonovy práce [6] mluvíme o začátku teorie informace, která položila potřebný základ pro další obory, jedním z nich je komprese dat. Informace je chápána jako veličina nezávislá na nosiči a na sémantickém významu. Informací se tak rozumí vlastnost vypovídat o jiné, přesně určitelné veličině. Jednotka této veličiny je pak odvozená podle logaritmického základu - pro základ 2 navrhl Shannon bit (Binary digit). Zároveň tato teorie definuje přenos informace jako matematickou transformaci vstupních zpráv na výstupní s přihlédnutím k prvku náhody (šumu).

Prvním problémem teorie informace pak je možnost redukovat výstupy z informačních zdrojů bez omezení informace v nich obsažených. Dalším problémem pak je hledání maximální propustnosti komunikačního kanálu a dosažení této kapacity. Roku 1950 pak Shannon přichází s prací, která naopak ukrývá informace tak, aby nebylo možné je získat bez klíče a tak položil základ dalšímu oboru, kryptografii.

Informace

podle ČSN 369001 význam, který člověk přikládá datům; zpráva. (více v [8])

Entropie informace

Míra neurčitosti před přijetím zprávy; množství informace. (více v [8])

Komprese dat

Snaha o zmenšení velikosti datových souborů - snižuje se entropie; postup, který se pro takové zmenšení velikosti datových souborů používá. (více v [9])

Ztrátová komprese dat

Postup komprese dat, který záměrně odstraní část informace, kterou pokládá za nedůležitou, přesné obnovení původní informace není možné. Tento způsob je hojně využíván při kompresi obrazu a zvuku. (více v [9])

Bezeztrátová komprese dat

Postup komprese dat, který umožňuje opětovné získání přesné vstupní informace. Limitem bezeztrátové komprese je entropie informace. (více v [9])

Kódování

Proces, při kterém jsou transformována zdrojová data na data komprimovaná. Opačný proces se nazývá **dekódování**. Při transformaci se používá kód - zobrazení, které přiřazuje každé zdrojové jednotce jedno kódové slovo (**kód**) a zároveň nepřizuje dvěma jednotkám stejná slova. Na kód máme i další požadavek: aby byl jednoznačně dekódovatelný. To splňují blokové kódy - kódy kde všechna kódová slova mají pevnou délku. **Prefixové kódy**, jsou kódy kde platí, že žádný kód není prefixem jiného a tak je zaručena jednoznačnost při čtení zleva doprava. Tyto kódy jsou také jednoznačně dekódovatelné, ale nemusí být blokové. (viz [4])

Redundance kódu

Rozdíl délky zprávy a entropie informace v ní obsažené. (viz [4])

Optimální kód

Kód s minimální redundancí. (viz [4])

Asymptoticky optimální kód

Kód, kde se délka kódového slova blíží entropii příslušné jednotky s rostoucí entropií zprávy. (viz [4])

Statistický model zdroje dat

Vstupní informace je tvořena množinou hodnot, vlastnosti této množiny můžeme popsat statistickým modelem. Pro zjednodušení je dále uvažována textová informace a množinou hodnot jsou znaky. (viz [4])

Statistický model nultého stupně

Každý znak textu je statisticky nezávislý na jiném znaku a pravděpodobnost výskytu každého znaku je stejná.

Statistický model prvního stupně

Výskyt znaku je sice nezávislý na výskytu jiného znaku, ale pravděpodobnosti výskytu jednotlivých znaků jsou různé.

Statistický model druhého stupně

Tento model zohledňuje různé pravděpodobnosti výskytu dvojice znaků v textu. Všem dvojicím je tedy přiřazena určitá pravděpodobnost.

Statistický model n-tého stupně

Sleduje se pravděpodobnost výskytu n-tice znaků v textu. Entropie informace klesá s rostoucím stupněm statistického modelu, například pro anglickou abecedu s mezerou je pro model nultého stupně entropie 4,75 bitů na znak a pro model 3. stupně 2,77 bitů na symbol.

3.2 Kódování celých čísel

V řadě kompresních algoritmů je třeba zakódovat celé číslo jednoznačně dekódovatelným kódem. Kromě blokového kódu můžeme použít i některý z prefixových kódů, například Fibonacciho kód.

Fibonacciho kódy¹

Kódy založené na Fibonacciho číslech řádu $m \geq 2$. Fibonacciho čísla jsou definována rekurzivním vztahem:

$$F_0 = F_{-1} = \dots = F_{-m+1} = 1$$

$$F_n = F_{n-m} + F_{n-m+1} + \dots + F_{n-1} \text{ pro } n \geq 1$$

Každé nezáporné číslo N má pak binární reprezentaci ve tvaru:

$$R(N) = \sum_{i=0}^k d_i F_i$$

kde $d_i \in \{0, 1\}$, $k \leq N$ a F_i ($0 \leq i \leq k$) jsou Fibonacciho čísla řádu 2. Vlastností této reprezentace je, že neobsahuje nikdy dvě jedničky za sebou. Fibonacciho kód řádu 2 je pak pro N

¹Převzato z [4]

vytvoříme reverzací $R(N)$ a doplnění o jedničku:

$$F(N) = d_0 d_1 d_2 \dots d_k 1$$

kde $d_i, (0 \leq i \leq k)$ vychází z $R(N)$. Výsledná binární slova tak tvoří prefixový kód, protože končí dvěma jedničkami, které se nikde jinde ve slově nevyskytují. Podobně se dají vytvořit i Fibonacciho kódy vyšších řádů, které komprimují ještě lépe. Přesto však žádný z těchto kódů není asymptoticky optimální. Protože jsou tyto kódy optimální do vysokého N , jsou použity i v demonstračních implementacích.

3.3 Další pojmy

Kompresní poměr

Poměr velikosti zakódovaných dat a původních dat.

Sourozenecká vlastnost

Binární strom má sourozeneckou vlastnost když:

1. každý uzel kromě kořene má sourozence
2. uzly mohou být seřazeny v pořadí neklesající pravděpodobnosti tak, že každý uzel sou sedící v seznamu s nějakým uzlem je jeho sourozenec (leví synové budou na sudých místech v seznamu a praví synové na lichých místech).

(Zpracováno podle [4])

Slovník

Dvojice $D = (M, C)$, kde M je konečná množina slov, C je zobrazení, které zobrazuje M na množinu kódových slov. $L(m)$ bude označovat délku kódového slova $C(m), m \in M$ v bitech. (Zpracováno podle [4])

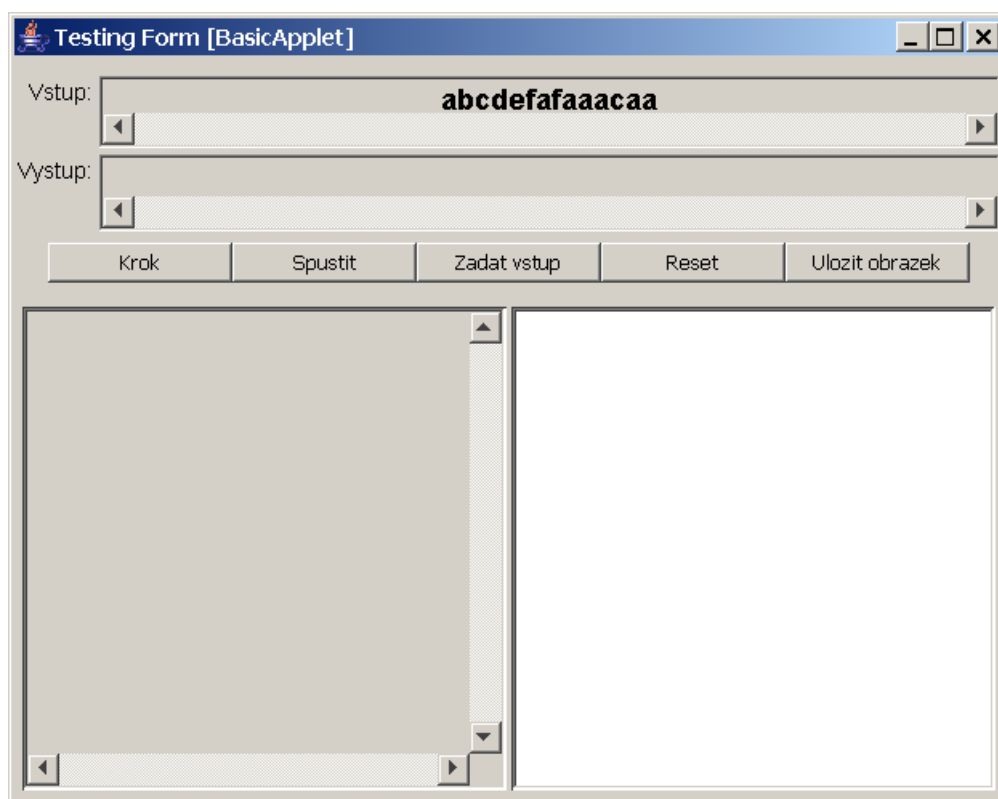
4 Návrh řešení

4.1 Použité technologie

Aplikace je řešena webovými stránkami, které obsahují informace o vybraném algoritmu a Java applet, který umožní sledování průběhu algoritmu na zadaném vstupu. Stránky jsou statické (bez potřeby serverových technologií), aby bylo možné je provozovat i lokálně - bez on-line přístupu na internet. Třídy obsahující kód appletů jsou zároveň spustitelné - pomocí Java Web Start je umožněno otevření aplikace přímo, bez nutnosti spouštění webového prohlížeče. Texty a popisky appletu jsou pro snadnou lokalizaci do jiných jazyků uloženy v `.properties` souborech. Jazyk je možné určit parametrem aplikace nebo appletu, bez parametru se použije systémové `Locales`. Applety potřebují ke svému běhu Javu 5.0. Pro vytvoření uživatelského rozhraní je použita platforma Swing, která je součástí klientských JRE a má dostatečné množství prvků pro tvorbu odpovídajícího rozhraní.

4.2 Uživatelské rozhraní

Třídy s výkonným kódem appletů jsou potomky nadřazené třídy, která vytváří základní uživatelské rozhraní a obsluhuje společné akce.



Obrázek 4.1: Základní uživatelské rozhraní

Základní GUI (viz obrázek 4.1) je rozdělené panely na části:

Vstup a výstup

Tento panel obsahuje dvě komponenty s posuvníky, které zobrazují vstup a výstup s možností zvýraznění části řetězce. Výstupní řetězec navíc může být rozdělen na dvě odlišené části - to je použito u algoritmů, které musí uložit do výstupu nejprve informace o vstupních datech a pak teprve zakódovaná data. V případě, že je vstup nebo výstup delší, než je možné zobrazit na šířku komponenty, jsou aktivní vodorovné posuvníky, které se navíc automaticky nastavují tak, aby byla vidět část vstupu-výstupu, se kterou se právě pracuje.

Ovládání appletu

Obsahem tohoto panelu jsou ovládací prvky (tlačítka) pro provedení kroku, spuštění automatického krokování, uvedení appletu do výchozího stavu a uložení grafického výstupu appletu ve formě obrázku.

Panel pro přídatné ovládání konkrétního appletu

Některé applety využívají tento panel ke zobrazení dodatečných ovládacích prvků, které ovlivňují vlastnosti konkrétního algoritmu.

Grafické zobrazení algoritmu

Tento panel slouží ke zobrazení grafického výstupu. Objekty grafického výstupu jsou potomky jedné třídy, takže je možné jejich uložení do typované kolekce. Navíc tato nadtřída bude poskytuje metody pro zjištění rozměrů grafického objektu a jeho umístění. To je nutné pro správnou funkci posuvníků grafického výstupu a také pro automatické oříznutí ukládaného grafického výstupu.

Textový zápis algoritmu se zvýrazněným krokováním

Textová komponenta se svislým posuvníkem a zalomením dlouhých řádků, která zobrazuje zadaný text a zvýrazní syntaxi definovaných slov. Zároveň umožňuje zvýraznění řádku kódu.

Informace o kompresi

Jednoduchý panel s popisky informujícími o velikosti vstupu v bitech, výstupu v bitech a kompresi. Hodnota velikosti výstupních dat je rozdělena na dvě položky - bity připadající na

uložení četností znaků ve vstupních dat a bity připadající na zakódování vstupních dat.

Rozdělení na panely je voleno tak, že panely s nejdůležitějším obsahem - algoritmus a grafické zobrazení mají přiděleny největší část a jsou umístěny vedle sebe - to vychází z předpokladu, že algoritmus je zapsán krátkými řádky a je tak možné ho v případě potřeby větší šířky pro zobrazení grafiky zúžit - delší řádky se případně zalomí, algoritmus je pak vidět celý, bez nutnosti použití posuvníků.

Grafické uspořádání je rozvrženo tak, aby bylo možné použití appletu v prohlížeči na monitoru s rozlišením 1024x768 a vyšším. Při zobrazení jako applet je velikost odvozena od dostupné velikosti stránky (100% výšky a 90% šířky), při použití na monitoru s menším rozlišením je třeba spustit aplikaci například prostřednictvím Java Web Start - zobrazovaná oblast pak není omezena ovládacími panely webového prohlížeče. Při samostatném spuštění aplikace je velikost libovolně měnitelná (od nastavené minimální velikosti výše) a změny velikosti se neprojevují pouze na prvcích s pevným obsahem. Ovládací prvky jsou umístěny blízko sobě, v jedné horizontální řadě. Jednotlivé akce jsou dosažitelné jediným kliknutím, prostředí navíc povoluje jen takové akce, které mají v daném stavu očekávatelný výsledek.

4.3 Grafický výstup

Grafické znázornění algoritmu vychází ze znázornění používaného v literatuře. Grafické prvky jsou navrženy tak, aby byly snadno čitelné a dostatečně demonstrující. Pro uložení grafického výstupu ve formátu EPS byla použita knihovna `org.jibble.epsgraphics`, která je pro nekomerční účely šířena pod licencí GNU General Public License. Tato knihovna obsahuje třídu `EpsGraphics`, jejíž instance může být jednoduše předána metodě `paint(Graphics g)`, která v Javě slouží k vykreslování grafiky. Objekt třídy `EpsGraphics` pak obsahuje postscriptové příkazy pro vykreslení grafiky. Třída navíc implementuje metody pro redukci barev na stupně šedi a nebo na pouze černobílou grafiku. Grafický výstup je možné uložit i v bitmapovém formátu PNG - tento výstup je implementován přímo v Javě v balíku `javax.imageio`.

4.4 Ovládání simulace

Simulace algoritmu probíhá v krocích, odpovídajících řádkům algoritmu, ale jeden řádek může trvat i více kroků. Kroky je možné spouštět jednotlivě pomocí tlačítka, nebo zvolit automatické spuštění, které bude v daných časových intervalech jednotlivé kroky provádět.

4.5 Implementace kompresního a dekompresního algoritmu v jazyce Java

Jazyk Java používá pro binární vstupně-výstupní operace zvláštní třídy odvozené od abstraktních tříd `java.io.OutputStream` a `java.io.InputStream` - streamy. Od těchto tříd jsou zděděny například třídy pro práci se soubory (`java.io.FileInputStream` a `java.io.FileOutputStream`), ale i speciální třídy `java.io.FilterInputStream` a `java.io.FilterOutputStream`, které jsou rodičovské pro takové streamy, kde se s daty během čtení nebo zápisu nějakým způsobem manipuluje a proto jsou třídy s implementacemi kompresních a dekompresních algoritmů zděděny právě z těchto tříd. Třídy `java.io.FilterInputStream` a `java.io.FilterOutputStream` také vhodně implementují abstraktní metody nadřazených tříd `java.io.OutputStream` a `java.io.InputStream`, takže pro vlastní implementaci kompresního algoritmu stačí pouze přetížení metod `write()+flush()` a pro implementaci dekompresního algoritmu přetížení metody `read()` při zachování plné kompatibility s jinými streamy - je možné je tedy bez omezení do sebe vnořovat tak, jak je v Javě obvyklé. Java obsahuje streamy pro práci s daty komprimovanými algoritmy GZIP a ZIP(jar).

5 Použité postupy

5.1 Webové stránky

Stránky jsou řešeny tak, aby byl jejich kód validní HTML 4.01 Transitional s využitím kaskádových stylů. Applety jsou do stránek vloženy pomocí tagu `<applet>`. Tento tag je sice v HTML 4.01 označený jako `deprecated`[7] a měl by být nahrazován obecným tagem `<object>`, ale webové prohlížeče při této náhradě reagují různě a v takovém případě je nutné použít skriptování (klientské nebo serverové) k detekci prohlížeče. Bohužel ani nástroj, který je pro tuto konverzi součástí Sun JDK - `htmlconverter` - nevygeneroval z kódu s tagem `<applet>` správně funkční náhradu, proto bylo kvůli udržení kompatibility mezi prohlížeči použito tagu `<applet>`. Protože je v aplikaci možnost uložení obrázku na disk, je applet podepsán - bezpečnostní mechanismy Javy neumožní uložení obrázku na disk bez přijetí podpisu. Jinak je ovšem applet funkční i bez přijetí podpisu. Dialog pro přijetí podpisu je vyvolán Javou při spouštění aplikace.

5.2 Grafický výstup

Všechny grafické objekty jsou potomky abstraktní třídy `Part`, která definuje abstraktní metodu `paintPart(Graphics g)` určenou k vykreslení objektu a metodu `getBounds()`, ze které je možné získat minimální obdélník opisující grafický prvek. Navíc obsahuje tato třída i metodu `fireSizeChangeEvent(SizeChange evt)`, která vytvoří událost `SizeChange` a odešle jí objektům, které jsou zaregistrovány k jejímu příjmu. K informaci o umístění objektu slouží parametry `X` a `Y` a pro manipulaci s grafickým objektem se pak použijí metody `moveTo(int x, int y)` pro absolutní nebo `move(int x, int y)` pro relativní posun. Tento návrh grafických tříd pak umožňuje jednoduchou práci s libovolně složitým grafickým zobrazením. To je využito ve třídě `DrawPanel`, která se stará o správné vykreslování. Všechny vykreslované objekty jsou uloženy v kolekci a k vykreslení pak stačí jednoduchá metoda:

```
public void paintArea(Graphics g, Rectangle area) {
    Graphics2D g2d = (Graphics2D) g;
    for (Part p:parts) {
        p.paintPart(g2d);
    }
    g2d.setClip(area);
}
```

Tato metoda je volána při metodě `paint`, parametr `area` je pak určen velikostí aktuálně zobrazené plochy panelu - používá se v případě, že má panel zobrazeny posuvníky. Pro uložení celého grafického výstupu do souboru se volá tato metoda s parametrem `area` nastaveným na hodnotu maximálních rozměrů vykreslených tvarů.

Podobně je ošetřena i reakce na událost `SizeChange`, kdy se spočítají rozměry všech komponent a pokud je potřeba, posunou se všechny grafické objekty tak, aby grafika začínala v levém horním rohu. Zároveň se nastaví rozměry panelu a tak je automaticky zaručena funkce posuvníků.

5.3 Spouštění pomocí Java Web Start

Pro snadné spuštění bez nutnosti použití webového prohlížeče je ve třídách s applety navíc implementována metoda `main`, která umožňuje spuštění třídy. V metodě `main` se vytvoří Swingové okno `JFrame` a jako jeho obsah se nastaví applet. Poté jsou zavolány metody, které inicializují třídu, když je spuštěna jako applet. Pro spuštění pomocí Web Start[5] je ještě třeba vytvořit speciální XML soubor, který popisuje chování aplikace při spuštění a umístění spouštěcí ikony v nabídce start (v MS Windows). Aplikace je nastavena tak, že je možné ji spouštět bez připojení k internetu. Mechanismus ochrany před zápisem na disk je podobný jako u appletu, proto se používají stejné podepsané soubory JAR. Při testování aplikace byla nalezena chyba Javy, který neumožní v případě, že uživatelský účet obsahuje speciální znaky, její spuštění pomocí Web Start. Java totiž ukládá JAR soubory spuštěné pomocí Web Start do složky v domovském adresáři uživatele, ale pokus získat pak informace z JAR jako při neexistenci souboru souboru pomocí `Resource.getBundle()` končí s výjimkou `MissingResourceException`. Řešením je použití uživatelského účtu, který neobsahuje speciální znaky.

5.4 Lokalizace

Veškeré popisky a texty v aplikaci jsou uloženy v souborech `.properties`, textových souborech, které obsahují dvojici `klíč=hodnota` a které se v Javě pro účely lokalizace používají. Pro čtení textů z těchto souborů je v aplikaci vytvořena třída `Localisation`, které je možné nastavit objekt `Locale` (pokud není nastaven, použije se nastavení systému) a třída pak podle tohoto objektu vybírá řetězce odpovídající zadaným klíčům metodou `getString(String key)`. Neexistující klíče jsou nahrazeny řetězcem oznamujícím chybu. Aktuální objekt `Locale` je možné z této třídy opět získat a použít například při formátování reálných čísel. Protože soubory `.properties` nemohou obsahovat diakritiku, jsou znaky s diakritikou převedeny do

Unicode sekvencí nástrojem `native2ascii` ze Sun JDK. Aplikace je dostupná v češtině a angličtině včetně odpovídajících překladů webových stránek. Výběr jazyka se provádí automaticky podle systémových `Locales`, nastavení jiného jazyka je možné u appletu nastavením parametru `locale` a u aplikace parametrem při spuštění ve formátu zápisu `Locales` (např. `cs_CZ`, nebo `en`).

5.5 Krokování algoritmu

Krokování algoritmu je realizováno tak, že pro algoritmus je vytvořen jednoduchý konečný automat a stav automatu je v programu realizován proměnou `state`, která je řídicí proměnou pro příkaz `switch`, který vybírá příslušnou část kódu v metodě `doNextStep()`, která je definována v rozhraní `Steped`. Toto rozhraní implementuje třída `BasicApplet` a tak i všechny applety. Volání metody `doNextStep()` pak následuje po stisknutí tlačítka pro další krok. Pro automatické krokování byla implementována jednoduchá třída zděděná z třídy `java.lang.Thread`. Toto vlákno pak jako parametr dostává objekt implementující rozhraní `Steped` a při každém zavolání metody `run()` zavolá metodu `doNextStep()` a uspí se na zvolenou dobu - experimentálně vybráno 100ms.

5.6 Statistický model dat

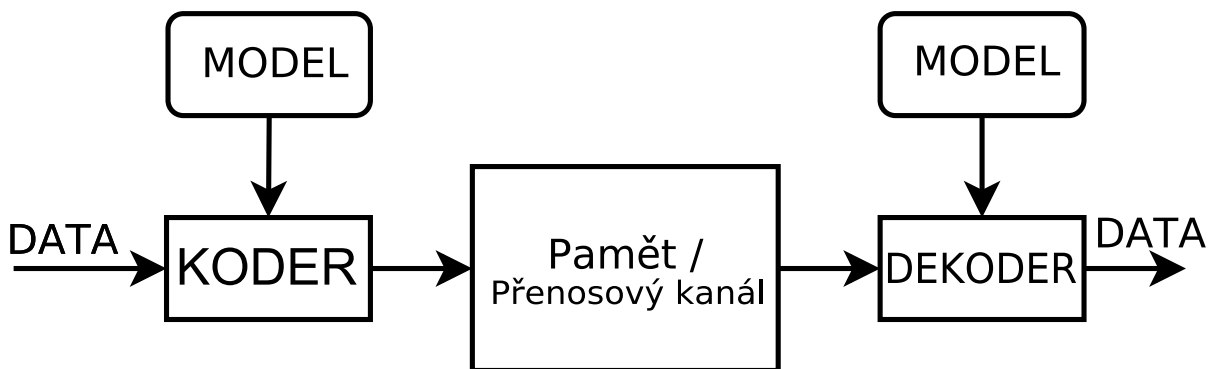
V některých představených algoritmech se pracuje se statistickým modelem dat prvního stupně. Ten vyžaduje informace o četnostech jednotlivých zdrojových jednotek a zároveň možnost jejich setřídění podle četnosti. Pro tyto jednotky byla vytvořena třída `CharInfo`, která udržuje informace o četnosti, přiřazený znaku a kódové slovo. Objekty `CharInfo` se porovnávají podle přiřazeného znaku, pro třídění podle četností je implementována třída `CharInfoCountComparator`. Pro implementaci algoritmů jako streamů byla vytvořena podobná třída `ByteInfo`.

5.7 Implementace kompresních a dekompresních algoritmů

Jak bylo uvedeno výše, součástí balíku jsou i třídy, které je možné použít libovolně s jinými streamy pro kompresi/dekompresi dat. Nevýhodou tohoto způsobu implementace je, že při zachování kompatibility je nutné stream pro kompresi, která vytváří statistický model, implementovat tak, že metoda `write()` pouze postupně vytváří statistický model a zapsaná data ukládá do dočasného souboru, odkud jsou přečtena po zavolání metody `flush()` a vytvoření kódů. Statistický model je ukládán jako hodnoty četnosti jednotlivých bytů zakódované Fibonacciho kódem řádu 2.

6 Statistické kompresní metody

Tyto metody jsou založené na statistickém modelu zdrojových dat. Kodér a dekodér musí mít k dispozici stejný model. Statistické modely bývají ovšem často podobné - například u textů a proto můžeme algoritmy podle schématu 6.1 rozdělit na:



Obrázek 6.1: Komprese s využitím statistického modelu

Statické kódování

Model pro kodér i dekodér je pevně určený bez ohledu na aktuálně kódovaná data. Tato metoda je vhodná pouze v případě, že kódujeme data, která mají podobné modely, například texty.

Semiadaptivní kódování

Nad kódovanými daty je nejprve vytvořen příslušný model, podle kterého jsou data zakódována. Tento model pak musí být odeslán spolu se zakódovanými daty. Výhodou je, že statistický model přesně odpovídá datům, nevýhodou je naopak to, že data se musí projít 2x - jednou pro vytvoření modelu a podruhé při kódování. Další nevýhodou je nutnost uložení statistického modelu do výstupu.

Adaptivní kódování

Model se vytváří během kódování ze vstupních dat a dekodér ho odvozuje z komprimovaných dat. Kódování i dekódování probíhá jedním průchodem, model dobře odpovídá komprimovaným datům.

6.1 Shannon-Fanovo kódování

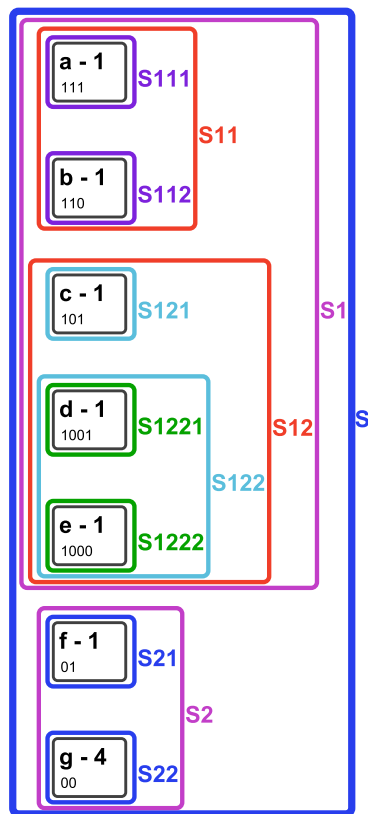
Tato metoda komprese dat byla vyvinuta v roce 1949 Claude Shannonem, Warrenem Weaverem a Robertem Fanem. Vychází ze statistického modelu prvního stupně - kódy jsou přiřazeny znakům na základě jejich četnosti bez ohledu na sousední znaky. Algoritmus postupně vytváří prefixový kód metodou shora dolů - kódová slova vznikají bit po bitu od prvního bitu. Principem je dělení na dvě podmnožiny o pokud možno stejné četnosti, které se rekurzivně provádí až do velikosti množiny 1. Tato metoda obecně nevytváří optimální kód. Algoritmus je založen na rekurzivním volání funkce split:

```
begin
    spočti četnosti zdrojových jednotek
    ulož četnosti do výstupu
    seříd je do neklesající posloupnosti
    SPLIT(S)
    vytvoř výstup
end

procedure SPLIT(S)
begin
    if |S|>1 then
        begin
            rozděl S na S1 a S2 s přibližně stejným počtem jednotek
            ke kódovým slovům S1 přidej 1
            ke kódovým slovům S2 přidej 0
            SPLIT(S1)
            SPLIT(S2)
        end
    end
end
```

Třída `gui.SFApplet` umožňuje simulaci semiadaptivní verze tohoto algoritmu. Statistický model zdrojových dat je uložen jako četnosti výskytu všech povolených znaků v abecedním pořadí. Četnosti jsou zvýšené o 1, aby byly zakódovány i znaky, které se nevyskytují, a jsou zakódovány Fibonacciho kódem 2. řádu a uloženy před kódovými slovy komprimovaných dat. Na obrázku 6.2 je vidět rozdělení na množiny v závěrečné fázi kódování řetězce `abcdefgggg` tak, jak ji poskytuje vizualizační applet. Kódy jednotlivých znaků jsou v tomto

okamžiku již plně určené. Řetězec `abcdefgggg` je zakódován tímto kódem do posloupnosti 27 bitů `111110101100110000100000000`. Statistický model se uvedeným způsobem zakóduje pro tento řetězec do 23 bitů.



Obrázek 6.2: Závěr Shannon-Fanova kódování

Implementace streamů pro kompresi a dekompresi jsou ve třídách `streams.SFOutputStream` a `streams.SFInputStream`.

6.2 Statické Huffmanovo kódování

Tato metoda vyvinutá roku 1951 Davidem Huffmanem poskytuje optimální kód při kódování podle statistického modelu 1.stupně. Kódování je založeno na konstrukci kódového stromu metodou zdola nahoru. Algoritmus semiadaptivní verze Huffmanova kódování:

`begin`

`spočti četnosti zdrojových jednotek`

`ulož četnosti do výstupu`

`setříd je do neklesající posloupnosti`

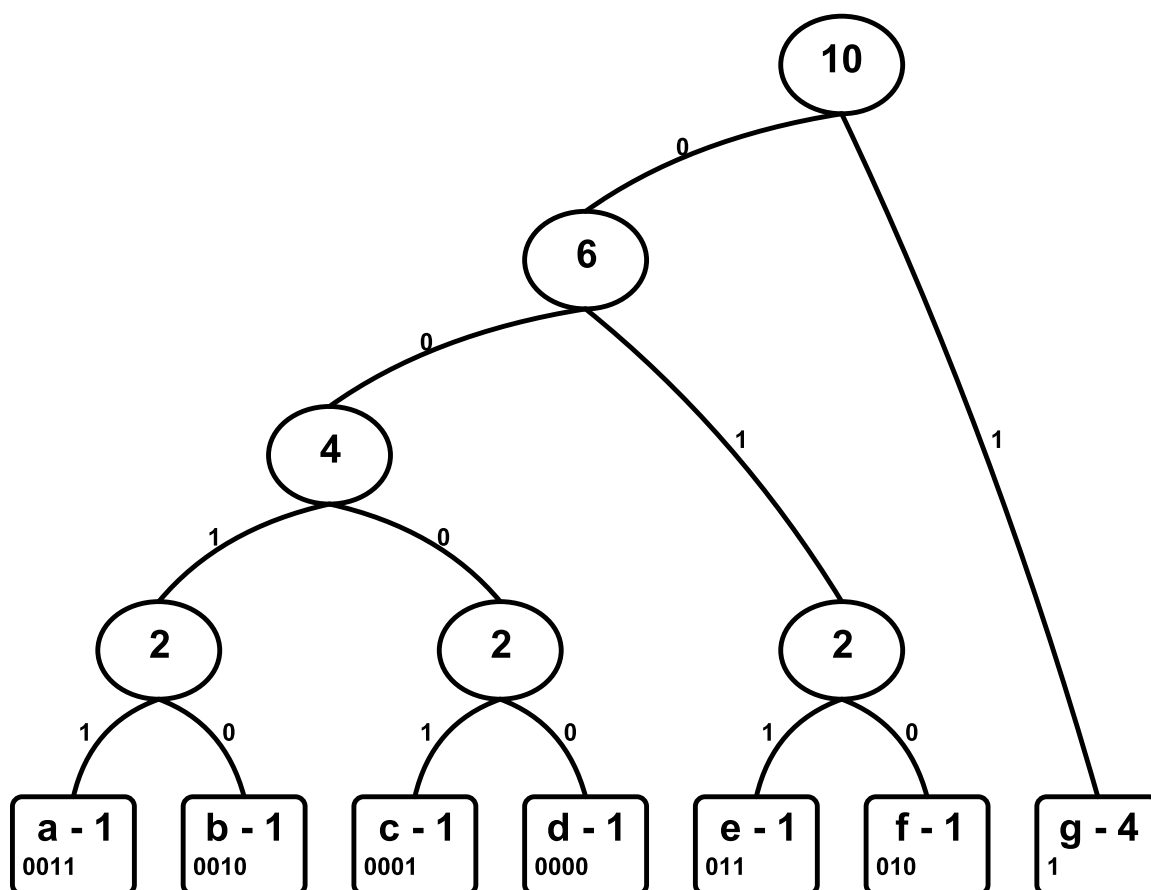
`vytvoř list stromu pro každou jednotku, vlož listy do fronty F`

```

while |F|>=2 do begin
    vyber z fronty dvě jednotky s nejnižším počtem
    vytvoř uzel ohodnocený součtem vybraných jednotek,
    následníci jsou vybrané jednotky
    zařaď nový uzel do fronty
end
list ohodnoť cestou z kořene do listu
vytvoř výstup
end

```

Tento algoritmus vytváří prefixový kód a kódový strom má sourozeneckou vlastnost. Takový strom se nazývá *Huffmanův*. Existuje-li více možností pro výběr dvojice s nejmenším ohodnocením z fronty, je možné vytvořit více Huffmanových kódů.



Obrázek 6.3: Závěr statického Huffmanova kódování

Třída SHApplet umožňuje simulaci tohoto algoritmu. K uložení statistického modelu je použit stejný postup jako u simulace Shannon-Fanova kódování - uložení četností Fibonacciho

kódem řádu 2. V případě, že je možné vytvoření více Huffmanových stromů, je vytvořen pouze ten, který je možné z počátečního seřazení listů vykreslit bez překřížení hran. Na obrázku 6.3 je vidět kódový strom pro řetězec `abcdefgggg` vytvořený appletem. Řetězec `abcdefgggg` je pak zakódován do posloupnosti 26 bitů `00110010000100000110101111`. Statistický model je zakódován stejně jako v předchozím případě do 23 bitů. Porovnáním s příkladem k Shannon-Fanovu kódování vidíme, že Shannon-Fanovo kódování netvoří optimální kód.

Implementace streamů pro kompresi a dekompresi jsou ve třídách `streams.SHOutputStream` a `streams.SHInputStream`.

6.3 Adaptivní Huffmanovo kódování

Nevýhody předchozího algoritmu byla nutnost průchodu zdrojových dat kvůli vytvoření statistického modelu a také nutnost uložení statistického modelu do výstupu. Tyto nedostatky je možné odstranit upravením na adaptivní verzi. Existuje více možností, jak vytvořit adaptivní verzi Huffmanova kódování, zde se však soustředíme na algoritmus FGK, vytvořený Fallerem, Gallagerem a Knuthem v roce 1985. Kódový strom se v tomto algoritmu tvoří postupně a data jsou kódována podle statistického modelu vytvořeného z dosud zakódované části. Po každém načtení znaku je třeba také provést aktualizaci stromu tak, aby byla zachována sourozenská vlastnost. Adaptivní algoritmy se také mohou lišit v tom, jak bude vypadat strom před začátkem algoritmu:

1. **Inicializace se všemi znaky abecedy** - v tomto případě je strom na začátku nastaven tak, že obsahuje všechny znaky se zvolenou pravděpodobností
2. **Použití uzlu ZERO** - v tomto případě obsahuje počáteční strom pouze jediný uzel ZERO. Při načtení znaku, který dosud nebyl zakódován, se do výstupu vypíše kód uzlu zero a uzel se rozdělí na nový uzel ZERO a list s novým znakem. Tento způsob je také použitý ve vizualizačním appletu.

Použitý algoritmus pro adaptivní kódování je následující:

```
begin
vytvoř uzel ZERO
načti_znak(X)
  while X!=EOF do begin
    if (prvníNačtení(X)) then
      begin
```



```

        do výstupu vypiš kód uzlu ZERO
        do výstupu vypiš X
        vytvoř nový uzel U s následníky ZERO a list s X
        aktualizuj_strom(U);
    end
else
    begin
        do výstupu vypiš kód uzlu s X
        aktualizuj_strom(uzel s X)
    end
    načti_znak(X)
end
end
end

```

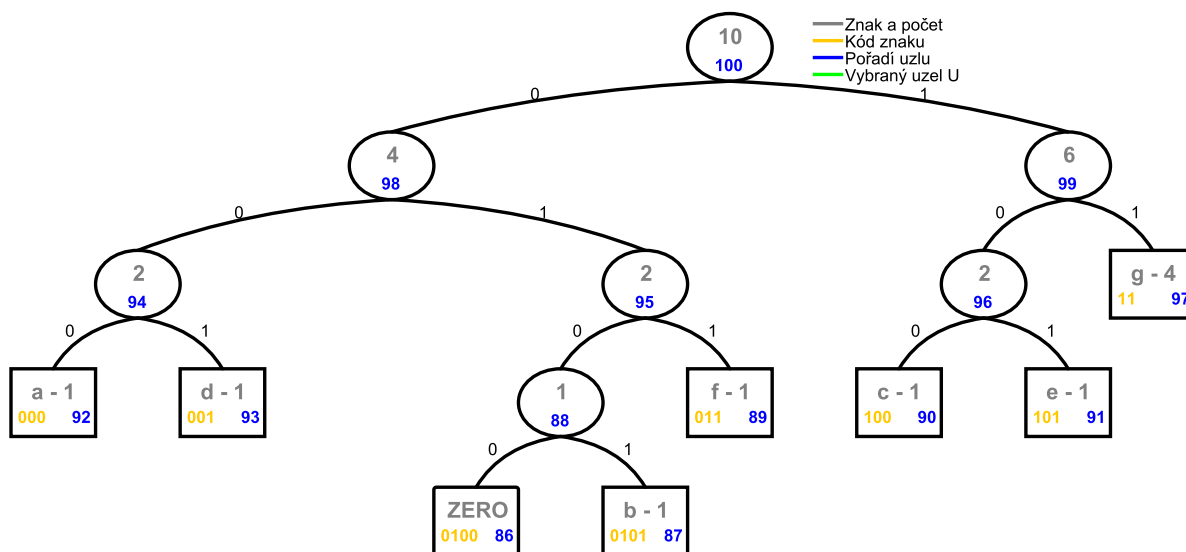
Funkce pro aktualizaci stromu je parametrizována uzlem, aby nebylo nutné upravovat celý strom, ale pouze vyjít z místa kde došlo ke změně a kontrolovat dodržení sourozenecké vlastnosti:

```

procedure aktualizuj_strom(U)
begin
    while (U!=kořen) do begin
        if (existuje uzel U1 se stejným ohodnocením a vyšším pořadím)
            then vyměň U1 a U
        zvyš ohodnocení U o 1
        U=předek(U)
    end
    zvyš ohodnocení U o 1, aktualizuj kódy listů
end

```

Vizualizace adaptivního Huffmanova kódování s aktualizací stromu podle FGK je dostupná ve třídě FGKApplet. Nové znaky se do stromu přidávají pomocí uzlu ZERO jak je popsáno výše. Zakódování řetězce `abcdefggggg` se pak pomocí tohoto algoritmu provede do řetězce `a0b00c100d000e1100f1000g01010011` o délce 46 bitů (při použití znaků z abecedy `abcdefg` zakódované do blokového kódu o délce 3 bitů). Závěrečný strom je vidět na obrázku 6.4.



Obrázek 6.4: Závěr adaptivního Huffmanova kódování

Vzhledem k tomu, že strom se během algoritmu mění, mění se i kódy jednotlivých znaků. Je tedy i možné, že různé znaky jsou během komprese zakódovány stejným kódem.

Implementace streamů kompresi a dekompresi FGK adaptivním Huffmanovým kódováním jsou ve třídách `streams.FGKOutputStream` a `streams.FGKInputStream`.

6.4 Aritmetické kódování

Huffmanovo kódování je speciálním případem aritmetického kódování. Výsledky rovné entropii dává Huffmanovo kódování pouze v případě, že pravděpodobnosti jsou rovné záporným mocninám dvou. V případě, že je rozložení jednotek nepříznivé, může Huffmanovo kódování pracovat s vysokou redundancí. To je způsobeno tím, že každý znak má přiřazen kód s celým počtem bitů.

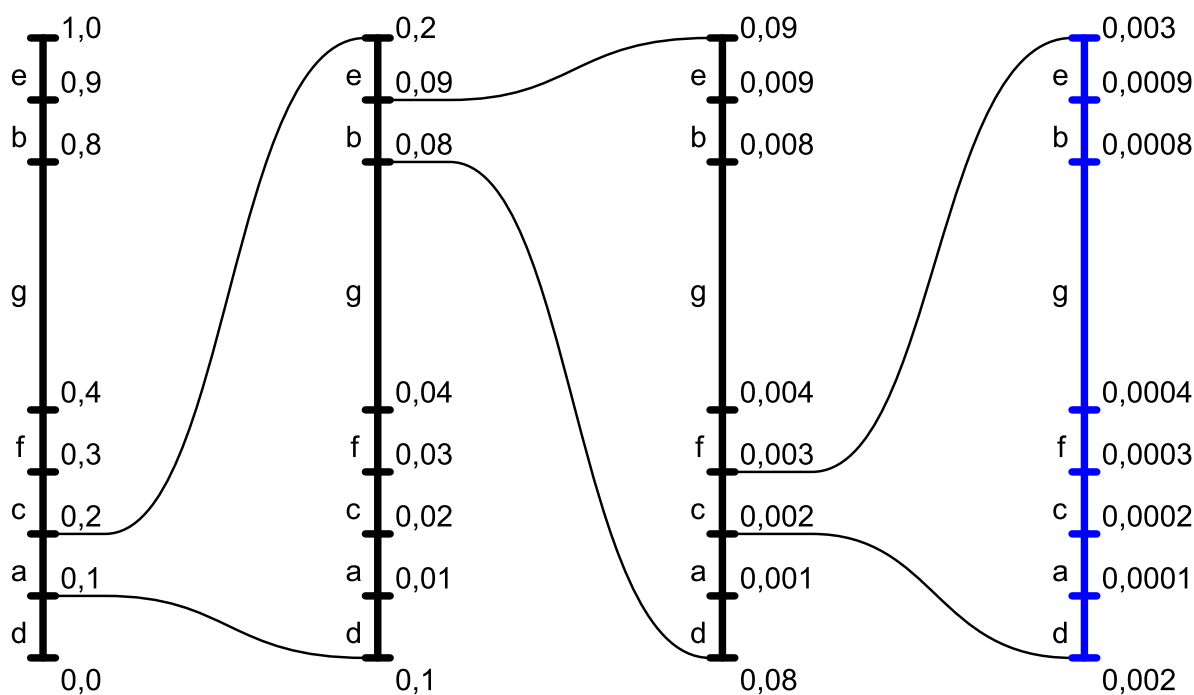
Aritmetické kódování oproti výše uvedeným metodám nekóduje jednotlivé znaky, ale rovnou celý kódovaný vstup do jediného čísla z intervalu $\langle 0, 1 \rangle$. To znamená že na znak může připadat necelý počet bitů. Toto kódování opět vychází ze statistického modelu prvního stupně. Každému znaku je pak přiřazena část intervalu $\langle 0, 1 \rangle$ a s načtením dalších znaků se interval postupně omezuje, jak je naznačeno na 6.5 pro znaky `abc` z řetězce `abcdefggggg`. Výsledkem je pak libovolné číslo z intervalu, který zbyl po načtení posledního znaku. Toto omezování intervalu sebou však přináší problém v podobě omezení práce s necelými čísly na počítačích. Algoritmus můžeme zapsat například takto:

```

begin
  spočti četnosti zdrojových jednotek
  interval I = nový interval 0..1
  rozděl I podle četnosti jednotek
  načti_znak(X)
  while X!=EOF do
    begin
      nové I = podinterval I odpovídající X
      rozděl I podle četnosti jednotek
      načti_znak(X)
    end
  výstup=libovolné číslo z intervalu I
end

```

Ve třídě `gui.ArithmeticApplet` je implementována vizualizace algoritmu, která pracuje s desetinnými čísly v dekadickém formátu.



Obrázek 6.5: Ukázka aritmetického kódování

Existuje řada implementací, které vylepšují práci s intervaly tak, aby se využívala pouze celočíselná aritmetika a omezená délka čísla. Také existuje hardwarová implementace algoritmu: Q-CODER vytvořený IBM.

Aritmetické kódování a různé jeho modifikace jsou chráněny mnoha patenty[1], dohromady jich je více než 50. To výrazně brání rozšíření těchto algoritmů do světa Open Source. Algoritmem patenty nezatíženým je Range encoding, které je koncipováno podle podobné myšlenky, ale kódování se provádí pouze v celočíselné aritmetice. Implementace aritmetického kódování pod Apache/BSD licencí je k dispozici v balíku `com.colloquial.arithcode`[2].

7 Slovníkové metody komprese dat

Metody založené na poznatku, že některé části se v datech opakují. Pro takové opakující se části je pak možné použít pouze odkaz do slovníku.

Podobně jako statistické metody můžeme slovníkové metody podle práce se slovníkem rozdělit na statické, semiadaptivní a adaptivní slovníkové metody.

Statické slovníkové metody

Používají ke kódování statický slovník, který je předem připraven. Pro přípravu tohoto slovníku se používají:

1. **Řetězce pevné délky - n-gramy** - například v angličtině se často opakují digramy „th“, „in“, „er“, „re“, „an“
2. **Řetězce proměnlivé délky** - tento způsob vytváří slovník pro nejfrekventovanější slova - v angličtině je 50% textu tvořeno asi 150 slovy.

Statické slovníkové metody nereagují na přesný výskyt slov v textu, proto dosažený kompresní poměr nemusí být dobrý.

Semiadaptivní slovníkové metody

Podobně jako u semiadaptivních statistických metod se i zde vytváří slovník ze zdrojových dat, ale poté musí být ke kódovaným datům přidán.

Adaptivní slovníkové metody

Tyto metody vycházejí z metod které popsali Lempel a Ziv ve dvou článcích v letech 1977 a 1978. Metoda LZ 77 používá posuvné okno, LZ 78 rostoucí slovník.

7.1 LZ 77

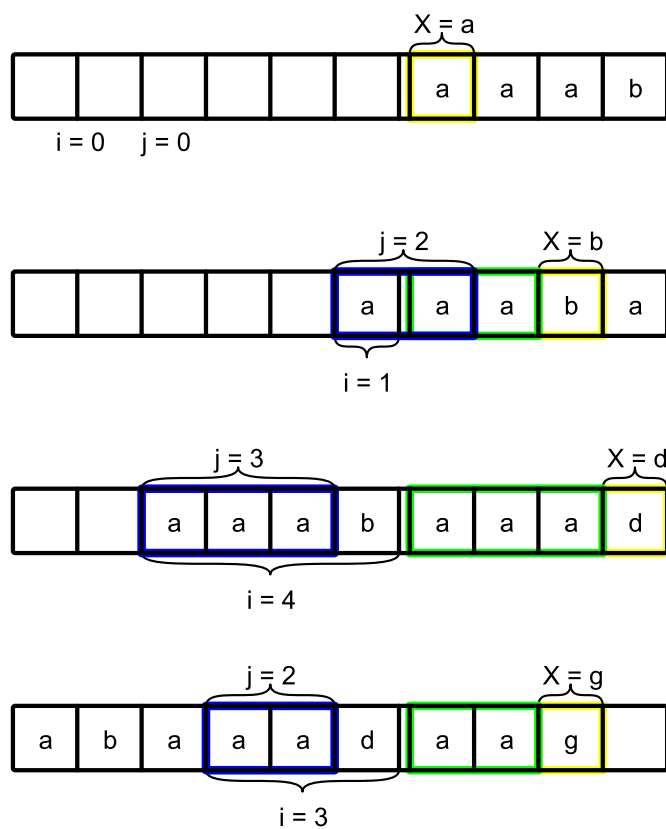
Původní metoda posuvného okna navržená v roce 1977 Abrahamem Lempelem a Jacobem Zivem. Tato metoda používá okno, rozdělené na zakódovanou a nezakódovanou část - výhled. Velikost zakódované části bývá $N \leq 8192$ a velikost výhledu bývá 10 až 20 bitů. Algoritmus pak vyhledá v nejdelší předponu výhledu, která začíná v zakódované části a zakóduje pak pomocí trojice (i, j, X) , kde i je vzdálenost předpony od hranice mezi nezakódovanou a zakódovanou

částí, j je délka nalezené předpony a X je první znak za předponou v nezakódované části. Algoritmus vychází z následujícího:

```

begin
naplň nezakódovanou část okna ze vstupu
while (nezakódovaná část není prázdná) do begin
    najdi v okně předponu  $p$  výhledu začínající v zakódované části
     $i$  = pozice předpony  $p$  v okně
     $j$  = délka předpony  $p$ 
     $X$  = první znak za  $p$  v nezakódované části
    do výstupu vypiš  $(i,j,X)$ 
    přidej do výhledu zprava  $j+1$  znaků
end
end

```



Obrázek 7.1: Kroky algoritmu LZ 77

Vizualizace algoritmu LZ77 je ve třídě `gui.LZ77Applet`. Ukázka kódování řetězce `aaabaaadaag` je vidět na obrázku 7.1. Výsledný výstup algoritmu je pak sekvence trojic $(0, 0, a)(1, 2, b)(4, 3, d)(3, 2, g)$, kterou je možné zakódovat do 40 bitů (při použití znaků z abecedy `abcdefg` zakódované do blokového kódu o délce 3 bitů, čísla zakódována Fibonacciho kódem řádu 2).

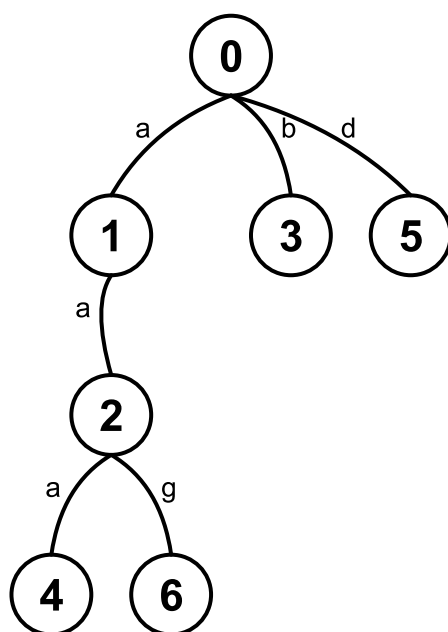
Také algoritmus LZ77 podléhá patentům[1]. Situace tohoto algoritmu je nepřehledná i v tom, že existuje celá řada patentovaných modifikací - jedna z nich byla dokonce patentována dvakrát, přestože se jednalo o stejný algoritmus. Patentovány jsou navíc také různé kombinace tohoto algoritmu s dalším kompresním algoritmem.

7.2 LZ 78

Metoda s rostoucím slovníkem z roku 1978, vytvořená Abrahamem Lempelem a Jacobem Zivem. V této metodě se text dělí na fráze, které se postupně přidávají do slovníku. Slovník může být realizován stromem, jak je ukázáno ve vizualizaci na obrázku 7.2. Problémem v něm může být neustále rostoucí slovník. Řešením je buď úplné vymazání slovníku při dosažení maximální velikosti, nebo ukončení vkládání do slovníku při dosažení maximální velikosti. Základní algoritmus tohoto kódování:

```
begin
w=\,,\‘‘
while (!EOF) begin
    načti_znak(X)
    if (wX ve slovníku) then
        w=wX
    else
        begin
            do výstupu vypiš (index(w),X)
            přidej X do stromu slovníku
            w=\,,\‘‘
        end
    end
end
end
```

Vizualizaci tohoto algoritmu je možné spustit ze třídy `gui.LZ78Applet`. Vizualizace zobrazuje postupné plnění slovníku a práci s ním. Na obrázku 7.2 je vidět slovník



Obrázek 7.2: Slovník při kódování LZ 78

po zakódování řetězce `aaabaaadaag`. Tento řetězec je zakódován do výstupní sekvence $(0, a) (1, a) (0, b) (2, a) (0, d) (2, g)$, která se s pomocí Fibonacciho kódu řádu 2 pro čísla a blokového kódu délky 3 pro znaky zakóduje do 35 bitů.

Také tento algoritmus je chráněn patenty[1], které brání jeho nekomerčnímu použití.

8 Závěr

Uvedené algoritmy a jejich modifikace můžeme najít v celé řadě programů, kde se pro dosažení větší efektivity navíc často kombinují tak, že data kódovaná jedním algoritmem jsou následně komprimována algoritmem jiným. Někdy je jsou také data před komprimací upravena, aby bylo dosaženo optimálních výsledků. Například u Open source algoritmu bzip2 se používá Huffmanovy komprese v kombinaci s Burrows-Wheelerovou transformací. Použití kompresních algoritmů také vyžadují některé formáty datových souborů. Komprese je také součástí datových přenosových protokolů, například na úrovni protokolu PPP určeného pro přenos dat pomocí modemů. Vzhledem k výrazně vysoké účinnosti kompresních algoritmů může být výrazným přínosem komprese na úrovni protoklu HTTP, kterou umí realizovat přímo webový server nebo jazyk, který generuje stránku (třeba PHP). Účelem této komprese není ani tak urychlení načítání stránky webovým prohlížečem, ale spíše úspora přenesených dat ze serveru. Například stránka `www.seznam.cz` posílaná v komprimované podobě má velikost cca 9 kB, velikost po dekomprimaci je 29kB. Za měsíc je to při více než 150 000 000 zobrazení hlavní stránky (viz [3]) úspora více než 3000 GB přenesených dat.

Aplikace, která je výsledkem této práce, poskytuje vizualizaci základních komprimačních algoritmů prostřednictvím HTML stránek s doprovodným textem a appletů s vizualizací. Aplikace je k dispozici v češtině a v angličtině. Součástí balíku jsou také implementace vybraných algoritmů jako streamů pro použití v Javě, ty mají ovšem vzhledem k tomu, že Java obsahuje výkonné komprimační algoritmy (GZIP, ZIP), spíše ukázkový charakter.

9 Seznam literatury

- [1] What about patents on data compression algorithms?, 2006.
- [2] B. Carpenter. Compression via arithmetic coding in java, 2005.
- [3] iAudit. Souhrnná zpráva o českém internetu za období 31.10.-27.11.2005, 2005.
- [4] B. Melichar. *Textové informační systémy*. ČVUT Praha, 1994.
- [5] S. Microsystems. Java web start overwiev, 2005.
- [6] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 a 623–656, 1948.
- [7] W. wide Web consortium. Objects, images, and applets in html documents, 1999.
- [8] Wikipedia. Informace - wikipedia, otevřená encyklopedie, 2005. [Online].
- [9] Wikipedia. Komprese dat - wikipedia, otevřená encyklopedie, 2005. [Online].

A Seznam použitých zkratek

JRE Java Runtime Environment

GUI Graphical User Interface

EPS Encapsulated PostScript

GNU GNU's Not Unix

PNG Portable Network Graphics

JDK Java Development Kit

XML Extensible Markup Language

JAR Java Archive

GZIP GNU zip

HTML HyperText Markup Language

B Obsah příloženého CD

B.1 Text diplomové práce

- `thesis/` - adresář s texty diplomové práce ve formátech PDF a PS
- `src/thesis.zip` - Zdrojové soubory pro diplomovou práci ve formátu \LaTeX

B.2 Aplikace

- `application/compression.jar` - podepsaný JAR archiv s přeloženými třídami pro vizualizaci
- `application/epsgraphics.jar` - JAR archiv s knihovnou `org.jibbble.epsgraphics` nutnou pro grafický výstup z aplikace
- `application/html/` - adresář s funkční verzí aplikace ve formátu webové prezentace s applety. Spuštění souborem `index.html`
- `application/html.zip` - archiv s funkční aplikací ve formátu webových stránek s applety
- `application/streams.jar` - JAR archiv s přeloženými třídami pro kompresi a dekompresi streamy
- `src/project.zip` - Zdrojové kódy aplikace včetně HTML stránek ve formátu projektu pro NetBeans 4.1