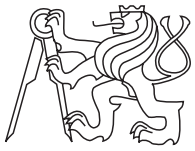
České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství
Obor: Aplikace softwarového inženýrství



Porovnání účinnosti komprese dat ve formátech XML a JSON

Comparison of the effectiveness of data compression in XML and JSON format

DIPLOMOVÁ PRÁCE

Vypracoval: Bc. Tomáš Smola
Vedoucí práce: Ing. Tomáš Liška, Ph.D.
Rok: 2015

Před svázáním místo téhle stránky vložte zadání práce s podpisem děkana (bude to jediný oboustranný list ve Vaší práci) !!!!

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne .....

.....
Bc. Tomáš Smola

Poděkování

Děkuji Ing. Tomáši Liškovi, Ph.D. za vedení mé diplomové práce a za podnětné návrhy, které ho obohatily.

Bc. Tomáš Smola

Název práce:
Porovnání účinnosti komprese dat ve formátech XML a JSON

Autor: Bc. Tomáš Smola

Obor: Aplikace softwarového inženýrství
Druh práce: Diplomová práce

Vedoucí práce: Ing. Tomáš Liška, Ph.D.
Katedra softwarového inženýrství, Fakulta jaderná a fyzikálně inženýrská, České vysoké učení technické v Praze
Konzultant: —

Abstrakt: Abstrakt

Klíčová slova: Klíčová slova

Title:
Comparison of the effectiveness of data compression in XML and JSON format

Author: Bc. Tomáš Smola

Abstract: Abstract

Key words: Key words

Obsah

Úvod

Závěrečnou diplomovou práci ke studijnímu oboru Aplikace softwarového inženýrství s názvem Porovnání účinnosti komprese dat ve formátu XML a JSON jsem si vybral z důvodu aktuálnosti – XML a JSON jsou v současnosti jedny z nejpoužívanější textových datových formátů – a také proto že toto téma velmi dobře propojuje teoretické znalosti získané při studiu s praktickými zkušenostmi v oboru softwarového inženýrství.

Dle výzkumu International Data Corporation (IDC) The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things [5] bylo pouze v roce 2014 vytvořeno a zkonzumovýno 2837 EB (exabytů) dat. Ze závěrů vyplývá, že se toto číslo každé dva roky zdvojnásobí, tedy v roce 2020 to bude již přibližně 40000 EB. Takové množství dat klade enormní požadavky na přenosové kanály a datová úložiště. Jako příklad mohou sloužit miliony shlédnutí oblíbených videí na serveru youtube.com. Pouze jedna sekunda videa v nekomprimovaném formátu CCIR 601 zabere více než 20 MB, tímto způsobem by zmiňovaná služba nemohla fungovat.

Vzhledem k tomu, že jsou technologie omezeny současnými možnostmi, znalostmi a také fyzikálními limity, je nutné hledat řešení jinde než v jejich zlepšování. Zde přichází na řadu komprese dat jako účinná metoda snížení velikosti objemu přenášených a ukládaných dat. Ve své práci bych čtenáře rád seznámil se základními principy komprese a vybranými kompresními algoritmy. Hlavním cílem je ale zodpovědět otázku, zda je možné dosáhnout dalších úspor volbou XML nebo JSON formátu a vhodného algoritmu, který využije znalosti struktury datového formátu.

Kapitola 1

Obecné seznámení s formáty XML a JSON

V této kapitole seznámím čtenáře se značkovacím jazykem XML a následně s JSONem, formátem pro výměnu dat. Mým cílem je popsat základní charakteristiky a syntaxi obou formátů tak, abych byl já, a následně i čtenář, schopen pochopit v kapitole 4 principy a výhody vybraných algoritmů využívajících znalosti struktury datových souborů.

1.1 XML

Na základě značkovacího jazyka SGML (Standard Generalized Markup Language), jehož obecnost činí úplnou implementaci velmi náročnou, vznikl vybráním nejpoužívanějších možností nový značkovací jazyk XML (eXtensible Markup Language), je tedy podmnožinou jazyka SGML. XML je obecný a otevřený, jeho vývoj a standardizaci realizovalo konsorcium W3C (World Wide Web Consorcium) [7]. XML umožňuje snadné vytváření konkrétních značkovacích jazyků pro popis dokumentů a dat ve standardizované, textově orientované podobě.

1.1.1 Charakteristika

V podstatě jde o textový dokument, jež je tvořen posloupností Unicode[1] znaků, ve kterém se rozlišují dva základní prvky: elementy (neboli značky) a obsah. Strukturu zapisovaných dat je možné vystihnout vzájemnému vnořování jednotlivých značek, které se ale nesmí křížit.

XML je pro člověka čitelné a spousta dnešních programů (například internetových prohlížečů) podporou zobrazení a vhodným formátováním tuto čitelnost ještě zvyšuje.

Jazyk XML se využívá hlavně pro publikování dokumentů, při výměně dat mezi různými systémy v prostředí internetu, jako univerzální úložiště dat či jako konfigurační soubor. Obecně lze říci, že je vhodný pro strukturovaná data. Do dokumentu můžeme vložit libovolná data, jejichž význam není bez použití schématu dokumentu zřejmý. Abychom mohli definovat sadu elementů (viz následující část 1.1.2), případně v nich i kontrolovat

[1] Unicode je standard pro konzistentní kódování, reprezentaci a manipulaci znaků většiny světových abeced.

typ dat, definujeme schéma dokumentu, které je vloženo buď přímo, nebo formou odkazu na definiční dokument.

1.1.2 Syntaktická analýza

Jak bylo řečeno již dříve, základními kameny XML jsou elementy a jejich obsah. Při práci s XML je nutné mít na paměti, že je na dodržení syntaxe kladen velmi velký důraz. Při dodržení správného způsobu zápisu a pravidel, která budou popsána níže, lze dokument považovat za tzv. well-formed XML [7].

Element

Základním prvkem každého XML dokumentu je element, který je vyznačen pomocí takzvaných tagů[2], mezi které může být vložen obsah. Počáteční i ukončující tag je dle definice [7] složen z dvojice znamének `<` (menší než) a `>` (větší než), mezi kterými je zapsán název tagu a volitelně i atributy. Ukončovací tag má navíc před svým názvem znak `/` (lomeno). Při správné aplikaci pravidel může vypadat element například následujícím způsobem:

```
<název_elementu název_atributu="hodnota atributu"></název_elementu>.
```

V případě, že element neobsahuje žádný obsah, lze ho zkráceně zapsat jako tzv. prázdný element:

```
<název_prázdného_elementu />.
```

V případě nedodržení správné syntaxe může nastat problém při rozpoznávání zapsaných dat, což může mít za následek nekompatibilitu mezi různými systémy při výměně dat.

Atribut

Počáteční tag elementu může obsahovat atributy upřesňující jeho význam. Atribut je vždy složen ze svého názvu a hodnoty, které jsou odděleny znakem `=` (rovná se). Hodnota je navíc zapsána mezi dvojici znaků `"` (uvozovky) nebo `'` (apostrof), přičemž hodnota může obsahovat jeden z těchto znaků tak, že se syntakticky nekříží. Následuje příklad atributu, jehož hodnota obsahuje znak `'`:

```
název_atributu="hodnota atributu osahující znak ' (apostrof)".
```

Obsah

Vše, co není tagem, je v dokumentu považováno za obsah. Kromě obyčejného textu mohou být obsahem další vnořené elementy, komentáře, instrukce pro zpracování, reference a další. Vzhledem k tomu, že určité znaky mají v syntaxi XML speciální význam (např. `<`, `>`), využívají se pro jejich zápis znakové entity[3]. Úplný výčet toho, co může XML dokument obsahovat, je včetně pravidel definován v [7].

[2] Tag definuje formu části textu.

[3] Pomocí znakových entit (sekvence znaků) lze zapsat znaky, které neobsahuje zvolená znaková sada, nebo mají v použitém kontextu speciální význam.

1.1.3 Vzorový příklad

Na následujících řádcích je zapsán XML element typu `Person`, oblíbená postavička Homer Simpson ze seriálu The Simpsons, který kromě jiného obsahuje vnořený element typu `Relatives`, který zde slouží jako kontejner s Homerovými příbuznými.

```
<Person>
 <FirstName>Homer</FirstName>
 <LastName>Simpson</LastName>
 <Relatives>
  <Relative>Grandpa</Relative>
  <Relative>Marge</Relative>
  <Relative>Bart</Relative>
  <Relative>Lisa</Relative>
  <Relative>Maggie</Relative>
 </Relatives>
</Person>
```

1.2 JSON

JSON neboli JavaScript Object Notation je odlehčený způsob zápisu (formátování) dat. Navzdory svému názvu jde o datově orientovaný textový formát nezávislý na počítačové platformě a čitelný pro člověka. Vznikl jako alternativa ke XML v oblasti výměny dat mezi systémy bez ohledu na použité technologie.

1.2.1 Charakteristika

Jak již název napovídá, je JSON velice úzce spojen s programovacím jazykem JavaScript – je na jeho syntaxi je založen. Hlavními prvky JSONu jsou dvě univerzální datové struktury: kolekce dvojic klíč/hodnota s unikátními klíči (tzv. associative array) a seřazený seznam hodnot (tzv. array), které podporují v nějaké formě asi všechny známé moderní programovací jazyky.

Silnou stránkou JSONu je rychlost zpracování JavaScriptem, bohužel právě přímé zpracování je zároveň potenciální hrozbou, neboť může být zneužito útočníkem k vykonání nějakého škodlivého kódu.

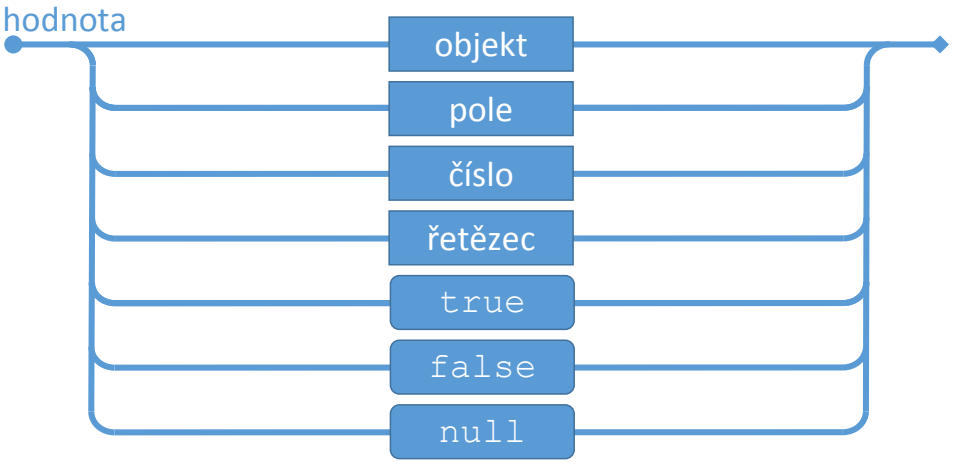
Orientace na data a zvolená syntaktická struktura, umožňují přehledný zápis datových objektů. Protože co jiného než než popis atributů (asociativní pole) a výčet hodnot (pole) datové objekty obsahují?

1.2.2 Syntaktická analýza

Dle definice [4] je JSON posloupností tokenů (viz 1.2.2) tvořených z Unicode znaků. Sada tokenů obsahuje šest strukturálních tokenů: `[` (levá hranatá závorka), { (levá složená závorka), `]` (pravá hranatá závorka), } (pravá složená závorka), `:` (dvojtečka) a `,` (čárka); dále obsahuje znakové řetězce, čísla a tři doslovné tokeny: `true`, `false` a `null`.

Hodnoty

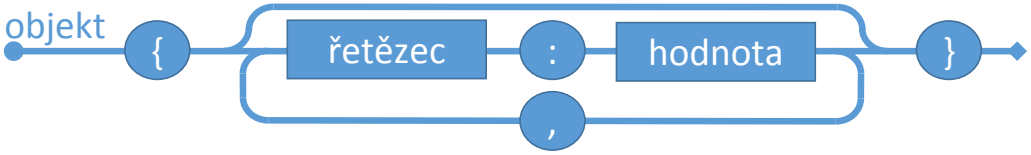
Za hodnotu je v JSONu považován objekt, pole, číslo, řetězec, `true`, `false`, nebo `null`.



Obrázek 1.1: Struktura hodnoty

Objekty

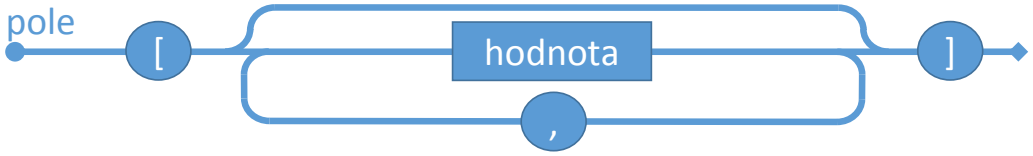
Objekt je reprezentován dvojicí složených závorek, uvnitř kterých je žádná nebo více dvojic klíč/hodnota, přičemž klíč je řetězec. Klíč a hodnota jsou odděleny dvojtečkou a jednotlivé dvojice odděluje čárka.



Obrázek 1.2: Struktura objektu

Pole

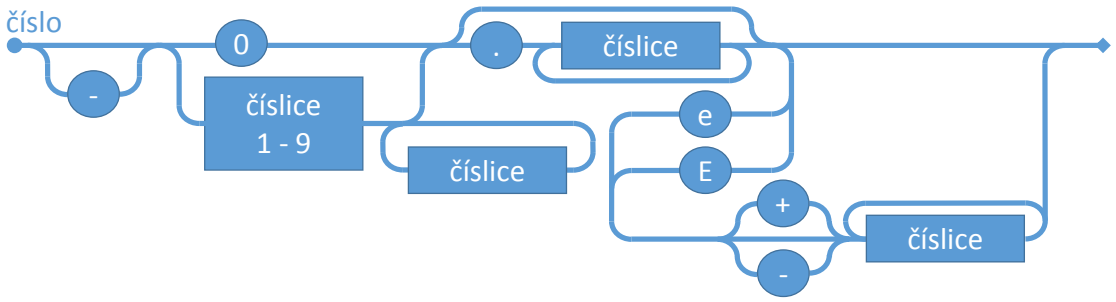
Pole je složeno z dvojice hranatých závorek, mezi kterými může být nula nebo více seřazených hodnot, které jsou odděleny čárkou.



Obrázek 1.3: Struktura pole

Čísla

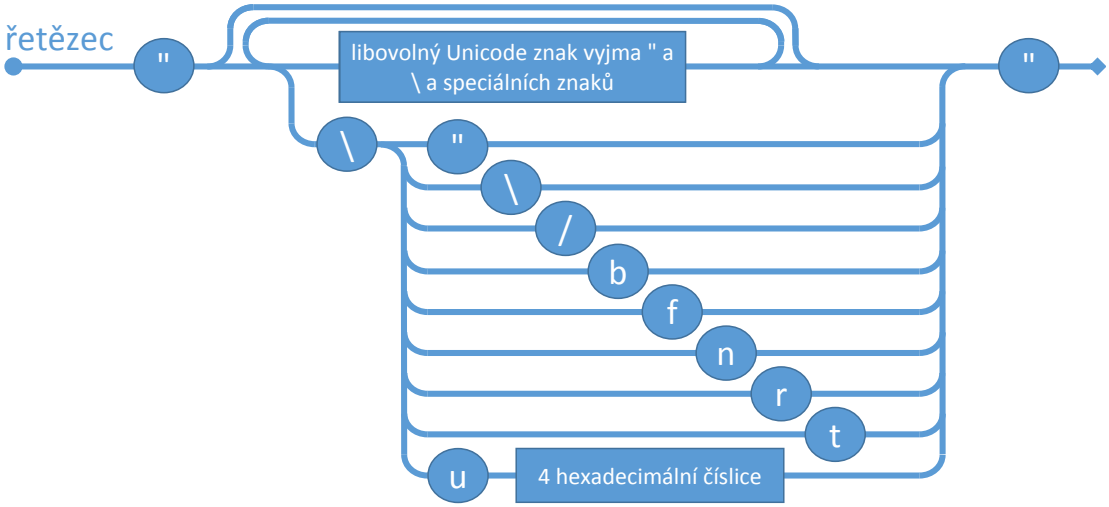
Čísla jsou v desítkové soustavě (tedy číslice $0-9$), záporná čísla jsou uvozena znaménkem `-` (mínus), desetinná část je oddělena znaménkem `.` (tečka). Je možný i takzvaný vědecký zápis čísel s použitím symbolů `e` (malé e) nebo `E` (velké e) a volitelně lze použít u exponentu znaménka `+` (plus) nebo `-` (mínus).



Obrázek 1.4: Struktura čísla

Řetězce

Řetězec je posloupnost Unicode znaků uvozená a zakončená znakem `"` (uvozovky). Mezi uvozovky mohou být zapsány všechny znaky kromě speciálních znaků, které jsou uvozeny tzv. únikovým znakem \ (zpětné lomítko), těmito znaky jsou například `"`, \, `t` (znak tabulátoru), `n` (znak posunu kurzoru na nový řádek) a `r` (znak posunu kurzoru na začátek řádku, známý jako návrat vozíku) a další. Kompletní výčet je možné vidět na obrázku 1.5 nebo v [4]. Navíc je možné každý znak zapsat pomocí kombinace `\u` a čtyřmístného hexadecimálního čísla odpovídajícího Unicode kódu požadovaného znaku. Řetězec obsahující pouze zpětné lomítko můžeme tedy zapsat následujícími způsoby: `"\\"`, `"\u005c"` nebo `"\u005C"` (u hexadecimálních čísel `A-F` nezáleží na velikosti).



Obrázek 1.5: Struktura řetězce

1.2.3 Vzorový příklad

Stejně jako ve vzorovém příkladě 1.1.3 ke XML, je zde popsána postavička Homera Simpsona včetně příbuzných, tentokrát ale v notaci JSON. Jde o objekt složený ze tří atributů, první dva mají hodnoty typu řetězec, k poslednímu `relatives` ale přísluší hodnota typu pole, ve kterém je vloženo pět příbuzných – hodnot typu řetězec.

```
{
 "firstName": "Homer",
 "lastName": "Simpson",
 "relatives": [
  "Grandpa",
  "Marge",
  "Bart",
  "Lisa",
  "Maggie"
 ]
}
```

1.3 Vzájemné porovnání XML a JSON

Je XML obecně lepší než JSON? Co vlastně dělá jeden formát lepší než jiný? Na toto bylo vedeno již nespočet diskusí. Argumentace obou stran diskuse je obvykle podložena jak teoretickou znalostí formátů, tak i praktickým použitím v reálných projektech. I přes to si myslím, že žádná diskuse nepřinesla jednoznačnou odpověď na výše položenou otázku, což samozřejmě pro některé konkrétní případy použití neplatí a volba jednoho z formátů je nutná nebo alespoň výhodnější. Ze svého zkoumání této problematiky si ale odnáším jeden podstatný závěr, který bych chtěl čtenáři zdůraznit: Vždy záleží na konkrétní situaci. Před samotným rozhodnutím, který z formátů využít, je vhodné zvážit důležité požadavky a do těchto úvah zahrnout klidně i další z existujících datových formátů. Zároveň lze ale říci, že dokud splňuje vybraný formát požadavky, tak ho lze považovat za dostatečný

Podíváme-li se znovu na vzorové příklady zápisu dat nebo položíme-li si vedle sebe stejná data zapsaná do obou formátů (viz příloha A.1), můžeme lehce vidět vzájemnou podobnost. Ve své práci proto vycházím z předpokladu, že jsou oba formáty z hlediska toho, jaká data lze do nich zapsat, ekvivalentní. Respektive budu pro další porovnání používat pouze takový druh dat, který bude tento předpoklad splňovat.

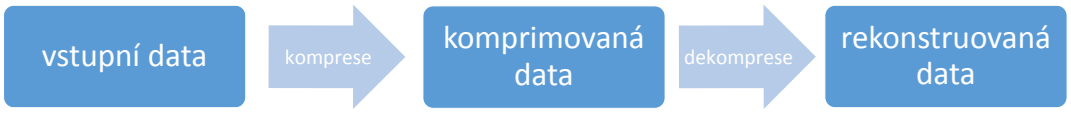
Kapitola 2

Komprese dat

Komprese nebo také komprimace dat je taková transformace dat, která má za cíl úsporu zdrojů při ukládání nebo archivaci a nebo snížení datového toku při přenosu, to vše při současném zachování informace obsažené v datech. Jinými slovy jde o redukci velikosti datových souborů, jehož následkem je úspora paměťových či přenosových kapacit. Postup, při kterém z komprimovaných dat rekonstruujeme data originální, se nazývá dekomprese.

2.1 Princip komprese dat

Data velmi často obsahují tzv. redundantní[1] informaci, toho právě využívá komprese – data jsou zpracována tak, aby byla redundance minimalizována. Jak lze vidět na obrázku 2.1, je na vstupní data použita operace komprese. Operací dekomprese dostaneme poté data rekonstruovaná – v závislosti na použité kompresní metodě, respektive na požadavcích získáme buď data přesně odpovídající původním a nebo pouze částečná. Z tohoto hlediska rozlišujeme dva typy kompresních metod: ztrátové a bezeztrátové.



Obrázek 2.1: Princip komprese

2.2 Typy kompresních metod

Jak název napovídá, při ztrátové kompresi ztratíme část informace obsaženou v původních datech, respektive jsou původní data pouze aproximována. Toto nám nemusí vadit například u obrázků, zvuku a videí, kde je využito nedokonalosti lidských smyslů. Lidské ucho nedokáže například slyšet velmi vysoké frekvence. Má smysl v datech určených k poslechu zachovávat informaci, kterou nemůže člověk slyšet? Častá odpověď je „ne“. Tohoto principu využívá mnoho kompresních metod, například známý zvukový formát MP3. Odstraněním „nepotřebné“ informace z dat je dosaženo ještě větší redukce objemu.

[1] Redundance znamená informační nadbytek, např. vícenásobný výskyt slov v textu.

Naopak v případě bezeztrátových metod je při kompresi zachována veškerá informace a při dekompresi jsou rekonstruována původní data. Těchto metod se využívá převážně tam, kde není možné původní data jakkoliv pozměnit. Například data ve formátech XML a JSON, kterým se věnuji v této práci, si nemůžeme dovolit pozměnit (přestanou mít původní význam), nebo dokonce ztratit.

2.3 Charakteristika komprese

Kompresní algoritmy lze hodnotit z mnoha různých úhlů pohledu. Můžeme měřit složitost algoritmu, rychlost, jakou jsou data komprimována a dekomprimována (to může být ovlivněno výkonem stroje, na kterém algoritmus běží), jak moc odpovídají rekonstruovaná data původním atd. Jednou z nejčastějších charakteristik je, logicky ze smyslu komprese vyplývající, tzv. kompresní poměr, který vyjadřuje velikost komprimovaných dat vůči původním, lze ho zapsat následujícím vztahem:

$$\texttt{kompresní poměr} = \frac{\texttt{délka původních dat}}{\texttt{délka komprimovaných dat}}. \tag{2.1}$$

Další sledovanou charakteristikou je tzv. úspora místa, která je vyjádřena jako:

$$\texttt{úspora místa} = \texttt{1 - kompresní poměr}^{-1}. \tag{2.2}$$

Mějme například 2D obrázek o velikosti 256×256 pixelů, který zabírá 65536 bytů. Obrázek zkomprimujeme a zabírá-li komprimovaná verze 16384 bytů, můžeme říct, že kompresní poměr je $4 : 1$ a úspora místa 75 %.

2.4 Míra informace v datech

Než se budeme moci věnovat jednotlivým kompresním metodám, připomeňme stručně myšlenky z teorie informace, které poskytly základ pro rozvoj metod bezeztrátové komprese. V této části předpokládám čtenářovu znalost základů teorie pravděpodobnosti a statistiky.

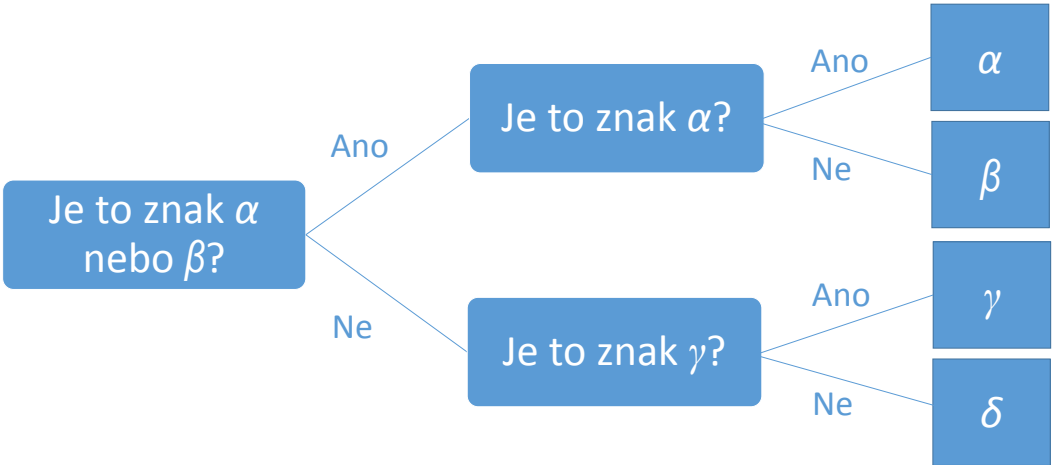
Informační entropie[2], jakožto míra neurčitosti v datech, nám pomůže lépe porozumět principům komprese. Představme si dva zdroje (označím je $\mathbf{Z}_1$ a $\mathbf{Z}_2$), které produkují zprávy složené ze symbolů (znaků) zdrojové abecedy $\mathbb{A} = \{\alpha, \beta, \gamma, \delta\}$. Oba zdroje generují znaky zprávy náhodně, pro zdroj $\mathbf{Z}_1$ však platí, že mají všechny znaky stejnou pravděpodobnost výskytu a to 25 %. Pro zdroj $\mathbf{Z}_2$ jsou pravděpodobnosti výskytu popsány v tabulce 2.1. Který ze zdrojů produkuje více informace?

Znak	Pravděpodobnost výskytu
α	$p_\alpha = 50$ %
β	$p_\beta = 30$ %
γ	$p_\gamma = 10$ %
δ	$p_\delta = 10$ %

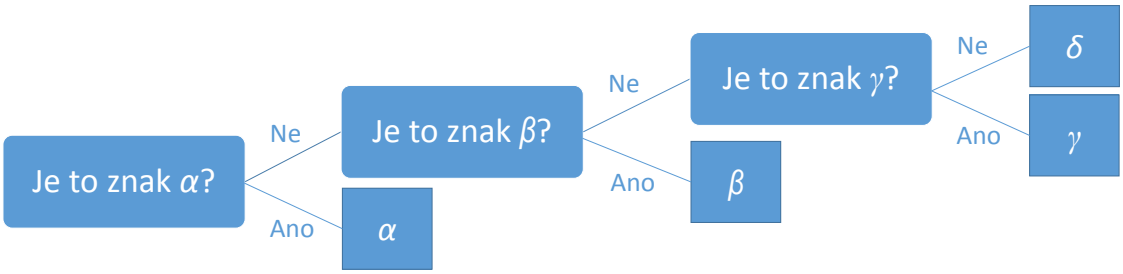
Tabulka 2.1: Pravděpodobnost výskytu znaků abecedy $\mathbb{A}$ ve zdroji $\mathbf{Z}_2$

[2]Nazývaná též Shannonova po Claudu E. Shannonovi, který zformuloval klíčové poznatky.

Claude E. Shannon si tuto otázku položil následujícím způsobem. Jaký je nejmenší počet otázek s odpověďmi ano nebo ne, které musíme zodpovědět, abychom byli schopni rozhodnout, jaký je následující znak každého zdroje? Odpovědí na tuto zdánlivě složitou otázku je vyhledávací algoritmus binární vyhledávání. Nejefektivnějším způsobem je v každém kroku položit takovou otázku, která rozdělí prohledávaný interval na dvě poloviny z hlediska pravděpodobnosti. Pro zobrazení, jaké otázky pokládat a v jakém pořadí, můžeme využít binární rozhodovací stromy (viz obrázky 2.2 pro zdroj $\mathbf{Z}_1$ a 2.3 pro zdroj $\mathbf{Z}_2$). Čteme je zleva doprava, v uzlech jsou otázky, hrany odpovídají odpovědím a v listech jsou příslušné znaky.



Obrázek 2.2: Binární rozhodovací strom pro zdroj $\mathbf{Z}_1$



Obrázek 2.3: Binární rozhodovací strom pro zdroj $\mathbf{Z}_2$

Průměrný počet otázek na zjištění jednoho znaku (tento počet označím $\bar{q}$) pokládaných tímto způsobem odpovídá pravděpodobnostem výskytu jednotlivých znaků a lze jej vypočítat jako $\bar{q} = \sum_{a \in \mathbb{A}} \#a \cdot p_a$, kde $\#a$ je počet otázek pro zjištění, že jde o znak a, a p_a je pravděpodobnost výskytu znaku a. Tedy pro zdroj $\mathbf{Z}_1$ dostaneme $\bar{q}_1 = 2p_\alpha + 2p_\beta + 2p_\gamma + 2p_\delta = 2 \cdot 0,25 + 2 \cdot 0,25 + 2 \cdot 0,25 + 2 \cdot 0,25 = 2$, tedy 2 otázky na znak, a pro zdroj $\mathbf{Z}_2$ dostaneme podobně $\bar{q}_2 = 1p_\alpha + 2p_\beta + 3p_\gamma + 3p_\delta = 1 \cdot 0,5 + 2 \cdot 0,3 + 3 \cdot 0,1 + 3 \cdot 0,1 = 1,7$, tedy 1,7 otázky na znak.

To znamená, že budeme-li se ptát na 1000 znaků u obou zdrojů, budeme muset položit 2000 otázek ke zjištění znaků ze zdroje $\mathbf{Z}_1$ a 1700 otázek ke zjištění znaků ze zdroje $\mathbf{Z}_2$. Výstup zdroje $\mathbf{Z}_2$ obsahuje méně překvapení, nebo lépe neurčitosti, než výstup zdroje $\mathbf{Z}_1$. Hovoříme o tom, že výstup zdroje $\mathbf{Z}_2$ poskytuje méně informace než výstup zdroje $\mathbf{Z}_1$.

2.4.1 Entropie informačního zdroje

V Shannonově teorii je definice zdroje informace prakticky totožná s definicí pravděpodobnostního prostoru, tj. trojice $(\mathcal{X}, \mathcal{S}, p)$, kde $\mathcal{X}$ je množina elementárních jevů (zpráv), $\mathcal{S}$ je třída náhodných jevů (σ-algebra podmnožin $\mathcal{X}$) a p je pravděpodobnostní míra na $\mathcal{S}$. [6]

Pro nás bude modelem zdroje informace diskrétní pravděpodobnostní prostor $(\mathcal{X}, p(x))$, který generuje zprávu X, jež je náhodnou veličinou s výběrovým prostorem $(\mathcal{X}, px(x))$. Mezi zprávou X a zdrojem $(\mathcal{X}, p(x))$ panuje ekvivalence, značíme $X \sim (\mathcal{X}, p(x))$. Základním pojmem teorie informace je entropie $H(X)$ zdroje X, jde o číslo, kterým charakterizujeme, jak obtížné je předpovědět hodnotu dané náhodné veličiny X, tj. zprávu, kterou vyprodukuje daný zdroj informace $X \sim (\mathcal{X}, p(x))$. Následuje formální definice entropie. [6]

Definice 2.1. *Entriopie zdroje informace* $X \sim (\mathcal{X}, p(x))$ *je veličina*

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x),$$

kde $0 \log 0 = \lim_{t \to 0_+} t \log t = 0$.

2.4.2 Entropie a komprese

Když chceme komprimovat digitální data, musíme je rozdělit na malé kousky, např. obrázek na pixely, textová data na znaky. To nám umožní zpracovat tato data jako posloupnost symbolů. Jednotlivé symboly pak můžeme reprezentovat s použitím nějakého kódu. Umíme-li zprávy vysílat a přijímat pouze v binární soustavě, vytvoříme kód[3] složený ze znaků kódové abecedy $\mathbb{B} = \{0, 1\}$.

Vraťme se nyní zpět k abecedě $\mathbb{A}$ a představme si zdroj $\mathbf{Z}$, který generuje zprávy složené ze znaků abecedy $\mathbb{A}$. Tento zdroj má pravděpodobnosti výskytu jednotlivých znaků odpovídající zdroji $\mathbf{Z}_2$, což my zatím ale nevíme. Jako příklad můžeme mít za úkol komprimovat text obsahující 1000 znaků vygenerovaných zdrojem $\mathbf{Z}$. V takovém případě intuitivně přiřadíme jednotlivým zdrojovým znakům nejkratší možná kódová slova viz kód κ_1 v tabulce 2.2. Tato slova mají délku 2, což znamená, že potřebujeme průměrně 2 bity na znak.

Zdrojový znak	Kódové slovo
α	11
β	10
γ	01
δ	00

Tabulka 2.2: Kód κ_1 pro zdroj $\mathbf{Z}_1$

Pokud analýzou zdroje $\mathbf{Z}$ ale zjistíme, že pravděpodobnosti výskytu odpovídají zdroji $\mathbf{Z}_2$ (viz tabulka 2.1), můžeme znakům přiřadit jiná kódová slova (viz tabulka 2.3), která budou brát ohled na nové pravděpodobnosti výskytu. Tento kód κ_2 je tzv. Huffmanův kód (viz kapitola 3 v [1]) a potřebuje průměrně 1,7 bitu na znak, způsob jeho vytvoření je popsán v části reference na Huffmanovo kódování nebo v [1].

[3] Takové binární kódy provází lidstvo již velice dlouho, např. kouřové signály, kódování znaků v Morseově abecedě.

Zdrojový znak	Kódové slovo
α	1
β	01
γ	000
δ	001

Tabulka 2.3: Kód κ_2 pro zdroj $\mathbf{Z}_2$

Podíváme-li se na obrázky 2.2 a 2.3 a příslušný výpočet průměrného počtu otázek na zjištění znaku, můžeme vidět jisté souvislosti. Například když slova Ano, resp. Ne na hranách rozhodovacího stromu nahradíme čísly 1, resp. 0, dostaneme čtením od kořene k listům odpovídající kódová slova. Stejně tak průměrný počet otázek na zjištění znaku odpovídá střední délce zvoleného kódování (viz definice 7 v [1]).

K jaké úspoře dojde? K zakódování 1000 znaků výše zmíněného textu pomocí kódu κ_1 potřebujeme 2000 bitů, ale při použití kódu κ_2 nám bude stačit pouze 1700 bitů. Úspora je tedy 300 bitů. Na otázku, zda je možné tuto úsporu ještě zvýšit, nelze odpovědět jenom ano nebo ne. Pokud zkrátíme délku některých kódových slov v kódu κ_2, stane se zakódovaná zpráva dvojznačnou a zpětně nedekódovatelnou. Kódujme například znak δ kódovým slovem 0, potom sekvence 01 může znamenat jak $\delta\alpha$, tak i β. Abychom odstranili tento problém, museli bychom mezi kódová slova vložit oddělovač, čímž ale zakódovanou zprávu prodloužíme a k požadované úspoře nedojde.

Již Claude E. Shannon tvrdil, že komprese má limit a tím je vždy entropie zdrojové zprávy. Čím je entropie nižší, tím vyšší je možnost komprese. Naopak čím je entropie vyšší, kvůli nepředvídatelnosti, schopnost komprese se snižuje. Pokud bychom chtěli komprimovat až za tento limit, museli bychom nutně část informace vypustit, což je ale přesně princip ztrátové komprese, kterou si nemůžeme vždy dovolit.

Kapitola 3

Statistické techniky komprese

Statistické kompresní metody využívají znalosti tzv. pravděpodobnostního modelu komprimovaných dat. Modelem může být například četnost výskytu jednotlivých symbolů v textu. Účinnost komprese je závislá na schopnosti co nejpřesněji modelovat zpracovávaná data. Čím více se model přibližuje realitě, tím je účinnost komprese vyšší. A naopak.

3.1 Pravděpodobnostní model

Pravděpodobnostní model přiřazuje pravděpodobnosti výskytu symbolům ve zdrojové zprávě. Na tomto základě například Huffmanovo kódování (viz 3.2) přiřazuje symbolům, které se v datech objevují častěji (mají vyšší pravděpodobnost výskytu), kratší kódová slova.

Podle toho, jakým způsobem model vzniká, ho můžeme označit jako statický, semiadaptivní, nebo adaptivní.

3.1.1 Statický model

Tento model je při implementaci kompresní metody a při použití se již nijak nemění. Výhodou je rychlost a jednoduchost vytvoření modelu. Například v případě, že budeme zpracovávat výstup pouze jednoho známého zdroje, se tímto způsobem vyhneme opakovanému vytváření modelu, které by vedlo ke stejným, nebo alespoň velmi podobným výsledkům. V opačném případě ale model nemusí datům vůbec odpovídat, čímž bude dosažená účinnost komprese velice nízká.

3.1.2 Semiadaptivní model

Komprimovaná data jsou jedním průchodem statisticky zpracována a následně je vytvořen odpovídající model. K samotné kompresi je ale nutný ještě druhý průchod daty s již vytvořeným modelem. Aby bylo možné data dekomprimovat, je nutné připojit model k datům. Model je sice přesný, ale velikost zkomprimované zprávy je větší o model, navíc přístup s dvěma průchody daty není efektivní.

3.1.3 Adaptivní model

Tento mode vzniká v průběhu zpracování dat a je postupně aktualizován. To platí i pro dekompresní algoritmus, ten vytváří model stejně jako kompresní algoritmus z části textu, kterou již dekomprimoval. Díky tomu není nutné model připojovat k datům a také procházet data dvakrát.

3.2 Huffmanovo kódování

Huffmanovo kódování je jednou z nejstarších kompresních technik, kterou publikoval již v roce 1952 David A. Huffman. Semiadaptivní dvouprůchodová verze byla od svého vzniku podrobena dalšímu výzkumu a postupně vylepšena až na adaptivní.

3.2.1 Semiadaptivní Huffmanovo kódování

Je založeno na dvou vlastnostech nejkratších prefixových (jednoznačně dekódovatelných) kódů [3]:

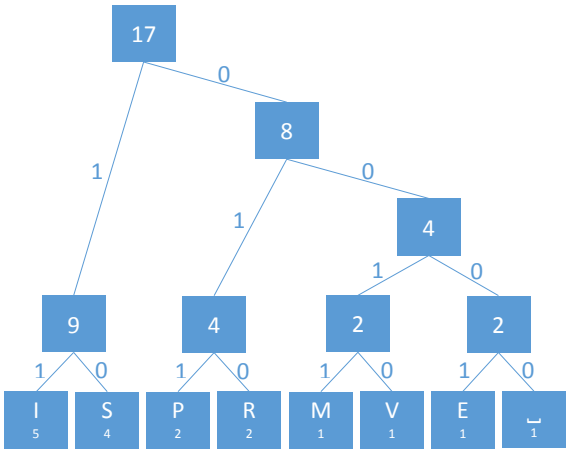
1. Symbolům s vyšší pravděpodobností výskytu jsou přiřazována kratší kódová slova.
2. Dvěma symbolům s nejmenší pravděpodobností výskytu jsou přiřazována kódová slova stejné délky.

Algoritmus nejprve seřadí symboly sestupně podle pravděpodobnosti výskytu a poté z nich v krocích konstruuje strom od listů ke kořeni. V každém kroku jsou vybrány dva symboly s nejmenší pravděpodobností výskytu, z nich je vytvořen nový uzel, který je brán jako symbol s kumulovanou pravděpodobností odpovídající součtu dílčích pravděpodobností. Kroky opakujeme až do vybrání všech symbolů, kdy je kumulovaná pravděpodobnost výskytu rovna 1. Nakonec ohodnotíme hrany vystupující z uzlu vlevo číslem 1 a vpravo číslem 0. Kódová slova získáme čtením stromu od kořene k listům. [2], [3]

Postup lze nejlépe prezentovat na příkladu. Mějme za úkol vytvořit Huffmanův kód pro zprávu „MISSISSIPPI RIVER“. Výsledný strom je zobrazen na obrázku 3.1, celá konstrukce pak v příloze reference na konstrukci stromu v příloze. Místo pravděpodobností výskytu jsou použity absolutní četnosti symbolů v listech a kumulativní četnosti v dalších uzlech. Získaná kódová slova pro jednotlivé symboly jsou zobrazena v tabulce

Zdrojový znak	Četnost výskytu	Kódové slovo
I	5	11
S	4	10
P	2	011
R	2	010
M	1	0011
V	1	0010
E	1	0001
␣	1	0000

Tabulka 3.1: Kódová slova semiadaptivního Huffmanova kódu



Obrázek 3.1: Strom vytvořený při semiadaptivním Huffmanově kódování

3.2.2 Adaptivní Huffmanovo kódování

Adaptivní Huffmanovo kódování konstruuje strom při kompresi i dekompresi posloupností stejných kroků, tzv. mirroring. Strom je v každém kroku upraven tak, aby produkoval nejkratší kód pro do té doby zpracovanou část dat. Jak se strom mění, mění se i kódy přiřazené jednotlivým symbolům. Listy stromu obsahují symboly a jejich četnosti výskytu, vnitřní uzly obsahují kumulovanou četnost svých potomků.

Implementovaných verzí existuje několik. Níže je představena verze, kterou navrhl Jeffrey S. Vitter a ve které všechny uzly navíc obsahují pořadové číslo, které je využito při rozhodování při aktualizaci stromu. Také se zde využívají tzv. bloky – množiny uzlů se stejnou četností.

Algoritmus začíná pouze s uzlem označeným NYT (not yet transmitted), který symbolizuje všechny znaky, které se zatím ve stromu nevyskytují. Je-li ke zpracování načten symbol, který se ještě ve stromě nevyskytuje, je do výstupu vypsáno kódové slovo uzlu NYT následované nezakódovaným a dohodnutým tvarem zpracovávaného symbolu. Následně se z uzly NYT stane vnitřní uzel a jsou mu přiřazeny dva noví potomci: nový uzel NYT a list reprezentující právě zpracovávaný znak. Poté je strom aktualizován. Je-li ke zpracování načten ve stromě již obsažený symbol, je na výstup vypsáno pouze kódové slovo tohoto symbolu a strom je aktualizován.

Nezakódovaný tvar zpracovávaného symbolu může být například jeho osmibitový ASCII kód. Má-li zdrojová abeceda $\{a_1, a_2, \ldots, a_m\}$ počet znaků roven m, pak zvolíme e, r tak, aby splňovala $m = 2^e + r$ a $0 \leq r < 2^e$. Znak a_k poté kódujeme jako $(e+1)$–bitovou reprezentaci čísla $k-1$, je-li $1 \leq k \leq 2r$, jinak je a_k kódován jako e–bitová reprezentace čísla $k-r-1$. [2], [3]

Pokusme se znovu zakódovat slovo MISSISSIPPI, kde budeme uvažovat anglickou abecedu jako zdrojovou. Ta má 26 znaků, tedy $m = 26$, $e = 4$ a $r = 10$ a znaky číslujeme $a_1 = A$, $a_2 = B$ atd. Postup konstrukce stromu je možné vidět v příloze na obrázku odkaz a způsob generování zprávy na diagramu odkaz. Začínáme se stromem, který obsahuje pouze kořen NYT. Čteme znak M a na výstup zapíšeme jeho pětibitový nezakódovaný tvar pro $k = 13$, tedy `01100`. Aktualizujeme strom. Čteme znak I, který se ve stromě také nevyskytuje. Na výstup zapíšeme kód uzlu NYT (nyní je to `0`) a pětibitový nezakódovaný tvar pro $k = 9$, tedy `01000`. Aktualizujeme strom. Dále čteme nový znak S. Na výstup zapíšeme

kód uzlu NYT (nyní je to `00`) a pětibitový nezakódovaný tvar pro $k = 19$, tedy `10010`. Aktualizujeme strom.

Výsledná zakódovaná zpráva je tvaru `01100|0|01000|00|10010|101|11|0|1|01|000|01111|1001|11`, kde jsem pro přehlednost vložil oddělovač „`|`“ oddělující jednotlivé významné bloky.

Kapitola 4

Přehled existujících implementací kompresních algoritmů pro efektivní uchovávání dat ve formátu XML a JSON

4.1 XML

4.2 Komprese JSONu

JSON byl navržen jako odlehčená varianta způsobu formátování dat proti XML, čímž byla odstraněna i redundance při použití počátečního a ukončovacího tagu. Po seznámení s jeho definicí a syntaxí je zřejmé, že zde již nezbývá moc prostoru k dalšímu „odlehčení“. Podíváme-li se ale na data zapsaná v tomto formátu, která mohou obsahovat například výstup SQL příkazu SELECT nad databází (viz odkaz na testovací soubor na přiloženém CD), můžeme vidět, že se zde některé „věci“ přeci jen opakují. Jsou to klíče, včetně uvozovek, v jednotlivých objektech, které stačí zapsat pouze jednou. Tohoto poznatku využívají i dva vybrané algoritmy JSONH a CJSON. Rád bych čtenáře upozornil, že algoritmů pro kompresi JSONu neexistuje mnoho, resp. jsou si velice podobné myšlenkou, provedením i názvy.

S daty v tomto formátu nepracujeme jako s obyčejným textem, ale převedeme je na kolekci instancí tříd, které reprezentují. V rychlosti tohoto zpracování vyniká především JavaScript.

4.2.1 JSONH

Tento algoritmus dovoluje komprimovat pouze homogenní kolekce dat, v terminologii JSONu jde o pole objektů, které mají stejný počet klíčů se stejnými názvy. Homogenitu dat musí zaručit uživatel, algoritmus samotný toto nikterak neošetřuje. Autoři projektu JSONH, kde je algoritmus implementován, na serveru github.com uvádějí, že data mohou být v některých případech zmenšena až na 30 %.

Vzorová data

Mějme homogenním poli zaměstnanců firmy, ve které máme uložené databázové id, příjmení a pozici, na které zaměstnanec pracuje. Tato kolekce může vypadat následujícím způsobem.

```
[
  { "id": 1, "name": "Sánchez", "position": "Manager" },
  { "id": 2, "name": "Duffy", "position": "Programmer" },
  { "id": 3, "name": "Tamburello", "position": "Worker" }
]
```

Postup komprese

1. Textová data jsou převedena na kolekci instancí tříd.
2. Z první instance v kolekci jsou určeny klíče a jejich počet.
3. Ze všech instancí v kolekci jsou postupně vyzvednuty hodnoty pro příslušné klíče, při čemž je zachováno pořadí klíčů, jak jsme jej získali v kroku 2.
4. Je vytvořen řetězec, resp. soubor jej obsahující, který je složen z počtu klíčů, seznam klíčů a hodnoty vyzvednuté v kroku 3.

Data po kompresi

```
[ 3, "id", "name", "position", 1, "Sánchez", "Manager", 2, "Duffy",
  "Programmer", 3, "Tamburello", "Worker" ]
```

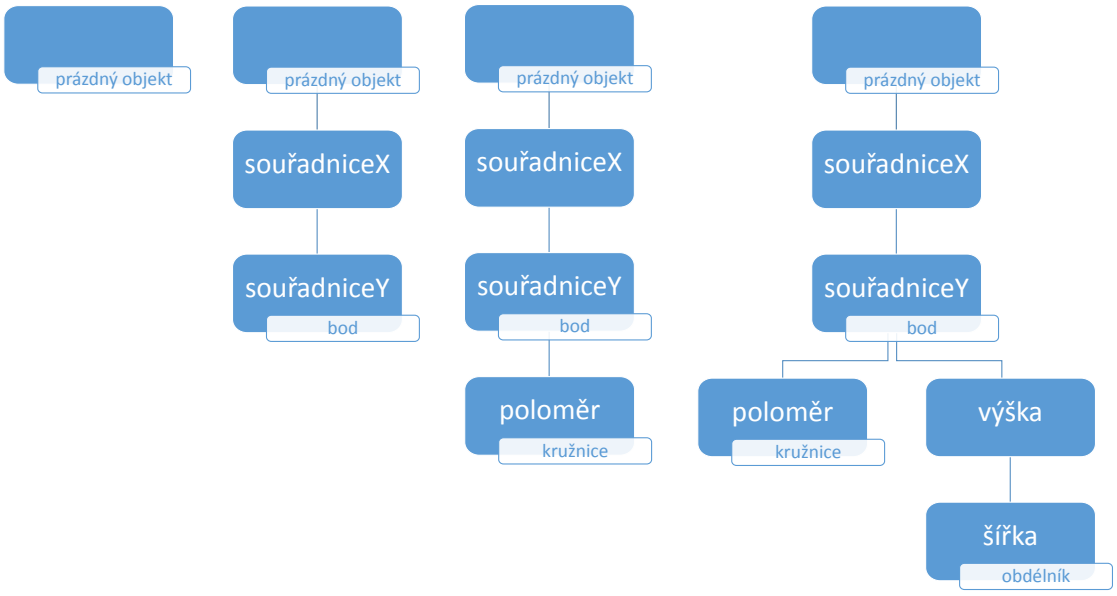
Postup rekonstrukce dat

1. Textová data jsou převedena na kolekci hodnot.
2. Z prvního prvku kolekce určíme počet klíčů, označíme n.
3. Z druhého až $(n + 1)$–ního prvku určíme klíče.
4. Z následujících prvků (po $n+1$) rekonstruujeme původní instance až do konce kolekce a ukládáme je do kolekce nové.
5. Novou kolekci serializujeme jako JSON do řetězce nebo souboru.

4.2.2 CJSON

Proti JSONH dokáže algoritmus CJSON komprimovat i nehomogenní data. K tomu, aby bylo při kompresi dosaženo významné úspory, je nutná určitá struktura dat. Tu bych přirovnal k dědičnosti, jak ji známe z principů objektově orientovaného programování. To znamená, že data lze popsat pomocí tříd, které sdílejí některé členy. Účinnost komprese závisí na poměru počtu tříd a množství dat k nim příslušných. Platí, že čím menší počet tříd a větší množství příslušných dat, tím větší je kompresní účinnost. Za ideál lze potom považovat homogenní data, tedy případ kdy stačí k popisu pouze jedna třída, ke které přísluší všechna data.

Algoritmus v průběhu komprese vytváří postupně strom šablon, který je na závěr vložen do výstupu ve formě jednotlivých šablon, která využívá již zmiňované dědičnosti. Konstrukce stromu je zobrazena na obrázku 4.1. Doplnit očíslování v obrázku a popis vytváření stromu.



Obrázek 4.1: Postup vytvoření stromu šablon

Vzorová data

Vytvořit data, popsat, upravit obrázek konstrukce stromu!!!

Mějme homogenním poli zaměstnanců firmy, ve které máme uložené databázové id, příjmení a pozici, na které zaměstnanec pracuje. Tato kolekce může vypadat následujícím způsobem.

```
[
  { "id": 1, "name": "Sánchez", "position": "Manager" },
  { "id": 2, "name": "Duffy", "position": "Programmer" },
  { "id": 3, "name": "Tamburello", "position": "Worker" }
]
```

Postup komprese

1. Textová data jsou převedena na kolekci instancí tříd.
2. Ze všech instancí v kolekci jsou postupně vyzvednuty hodnoty a uloženy do kolekce objektů, které obsahují pouze kolekci příslušných hodnot, zároveň je konstruován strom šablon.
3. Ze stromu šablon je vytvořena kolekce šablon. Šablona je kolekce obsahující identifikátor šablony, ze které „dědí“, a klíče. Identifikátor 0 je rezervován pro šablonu odpovídající prázdnému objektu.
4. Objekty s hodnotami jsou doplněny o identifikátor odpovídající šablony.

5. Ze vzniklých kolekci vytvoříme objekt obsahující identifikátor kompresního algoritmu (klíč `"f"`), kolekce šablon (klíč `"t"`) a kolekce hodnot (klíč `"v"`) (viz Data po kompresi).

6. Objekt vzniklý v bodě 5 serializujeme jako JSON do řetězce nebo souboru.

Data po kompresi

```
{ "f": "cjson",
  "t": [ [ 0, "x", "y"], [ 1, "width", "height"] ],
  "v": [ { "": [ 1, 100, 100 ] }, { "": [ 2, 100, 100, 200, 150 ] } ] }
```

Postup rekonstrukce dat

1. Textová data jsou převedena na instanci třídy.

2. Vytvoříme kolekci, do které postupně vkládáme instance tříd vzniklé z prvků kolekce hodnot a příslušných šablon.

3. Tuto kolekci serializujeme jako JSON do řetězce nebo souboru.

Kapitola 5

Vlastní implementace vybraných kompresních algoritmů

Kapitola 6

Porovnání účinnosti komprese dat ve formátu XML a JSON

Závěr

Seznam použitých zdrojů

[1] MAREŠ, Jan. *Teorie kódování.* 1. vyd. Praha: Česká technika, 2008. 120 s. ISBN 978-80-01-04203-8.

[2] SALOMON, David. *Data Compression: the complete reference.* 4th ed. London: Springer, 2007, xxv, 1092 s. ISBN 978-1-84628-602-5.

[3] SAYOOD, Khalid. *Introduction to data compression: the complete reference.* 4th ed. Waltham, MA: Morgan Kaufmann, c2012, xvi, 740 s. ISBN 978-0-12-415796-5.

[4] Standard ECMA-404. *The JSON Data Interchange Format.* Geneva: Ecma International, 2013, [online], [cit. 28. října 2014]. Dostupné na: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>.

[5] The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things. INTERNATIONAL DATA CORPORATION. *EMC* [online]. 2014 [cit. 2015-02-25]. Dostupné z: <http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm>.

[6] VAJDA, Igor. *Teorie informace.* Vyd. 1. Praha: Vydavatelství ČVUT, 2004. 109 s. ISBN 80-01-02986-7.

[7] W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition).* 26. listopadu 2008, [online], [cit. 26. listopadu 2014]. Dostupné na: <http://www.w3.org/TR/2008/REC-xml-20081126/>.

Přílohy

Příloha A

Název/obsah přílohy

A.1 Porovnání XML a JSON

XML

```
<widget>
 <debug>on</debug>
 <window
  title="Sample Widget">
  <name>main_window</name>
  <width>500</width>
  <height>500</height>
 </window>
 <image
  src="Images/Sun.png"
  name="sun1">
  <hOffset>250</hOffset>
  <vOffset>250</vOffset>
  <alignment>center</alignment>
 </image>
 <text
  data="Click Here"
  size="36"
  style="bold">
  <name>text1</name>
  <hOffset>250</hOffset>
  <vOffset>100</vOffset>
  <alignment>center</alignment>
  <onMouseUp>sun1.opacity =
   (sun1.opacity / 100) * 90;
  </onMouseUp>
 </text>
</widget>
```

JSON

```
{ "widget": {
 "debug": "on",
 "window": {
  "title": "Sample Widget",
  "name": "main_window",
  "width": 500,
  "height": 500
 },
 "image": {
  "src": "Images/Sun.png",
  "name": "sun1",
  "hOffset": 250,
  "vOffset": 250,
  "alignment": "center"
 },
 "text": {
  "data": "Click Here",
  "size": 36,
  "style": "bold",
  "name": "text1",
  "hOffset": 250,
  "vOffset": 100,
  "alignment": "center",
  "onMouseUp": "sun1.opacity =
   (sun1.opacity / 100) * 90;"

 }
}}
```