Projekt z předmětu Kryptografie a počítačová bezpečnost

Téma: Prefixové kódy

VŠB-TU Ostrava:Fakulta Elektrotechniky a informatiky

březen 2011

Martin Dočkal
doc068
dockal.martin@gmail.com

1 Obsah

2 Abstrakt

Následující práce se zabývá jedné z mnoha oblastí počítačové kryptografie a bezpečnosti - prefixovému kódování a implementaci vybraných algoritmů, které našly uplatnění zejména s příchodem zavádění číslicové digitální techniky v 50. letech. Také samotná oblast prefixového šifrování je velmi obsáhlá a rozmanitá, řada algoritmů a studií sebou nese mnoho dalších variací od stejného autora nebo autorů, proto i zde si popíšeme pouze jejich společnou charakteristiku a představíme si průkopnické algoritmy této doménové oblasti v jejich základním (prvotním) kontextu.

3 Klíčová slova

Prefixový kód, prefix, komprese, Huffmann, Shannon, algoritmus, dešifrování, binární strom, substituce

4 Seznam použitých symbolů a zkratek

RLE ..... Running length encoding

5 Úvod k prefixovým kódům

Společnou charakteristikou všech prefixových algoritmů je nahrazení každého vstupního znaku otevřené abecedy vždy stejným příslušným řetězcem abecedy šifrové o délce vždy alespoň rovno 1, který není prefixem (předponou) žádného jiného prefixu. Tuto na první pohled obtížnou formulaci nesoucí jednoduchou myšlenku se pokusím vysvětlit na prvním demonstračním příkladě formou tabulky (Tabulka 1).

Znak otevřené abecedy	Prefixový kód
A	1
B	01
C	00

Tabulka 1 : Úvodní příklad k prefixovým kódům

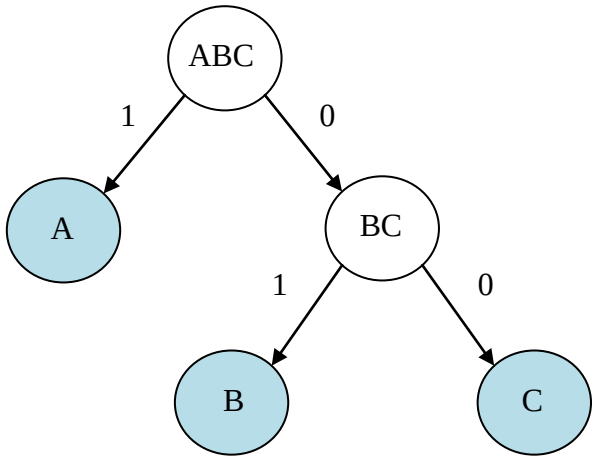
Z tabulky a uvedené formulace prefixových kódů je zřejmé, že v našem případě nemůže nastat situace, že by došlo k nejednoznačnosti v přiřazování prefixového kódu šifrové abecedy ke znaku abecedy otevřené při šifrování, a také při přiřazování znaku otevřené abecedy znaku šifrové abecedy při dešifrování (tzv. zpětném směru). Z matematického hlediska se jedná tedy o zobrazení z m na n. Nad tabulkou, která nám pouze říká, jaká jsou pravidla a přiřazeni prefixových kódů symbolům šifrové abecedy, si vyzkoušíme šifrování nad nějakou posloupností symbolů otevřené abecedy:
Uvažujme následující slovo :

```
CBAABB
```

Podle tabulky jednoduše provedeme substituci, kde výsledek bude mít podobu :

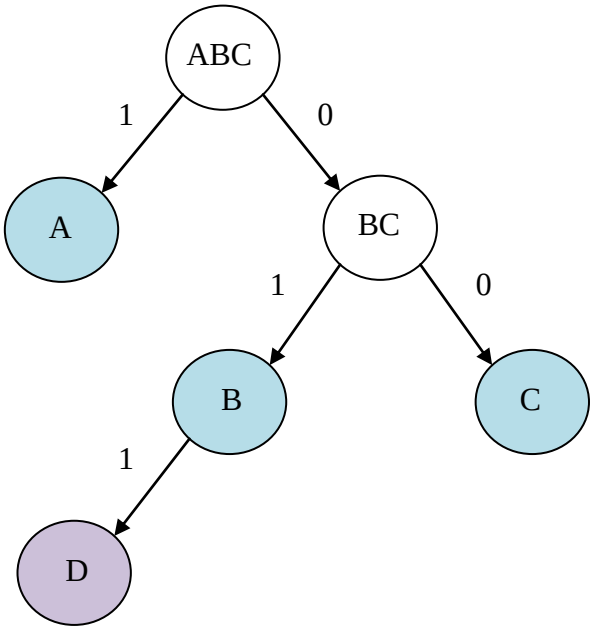
```
0001110101
```

Při dešifrování se provádí substituce opačná, kdy čteme postupně takto vytvořenou posloupnost šifrové abecedy zleva doprava symbol po symbolu, dokud nenarazíme na takový symbol, který spolu s předchozími tvoří prefix nějakého znaku otevřené abecedy. Pokud ano, zapíšeme podle tabulky příslušný znak otevřené abecedy na výstup, dosud přečtenou sekvenci můžeme již uvolnit z čtecí paměti a další znak přečtený znak šifrové abecedy v posloupnosti již ukládáme do paměti prázdné, takto celý cyklus opakujeme až do konce zašifrované posloupnosti.
Překladovou tabulku za účelem snadnějšího a rychlejšího procesu šifrování, nebo dešifrování lze velmi dobře zachytit binárním stromem, viz Obrázek 1.



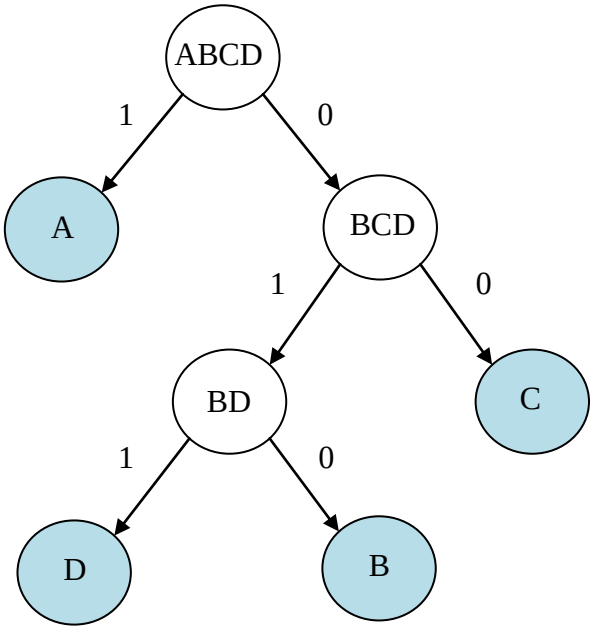
Obrázek 1: Binární strom zachycující situaci překladové tabulky ukázkového příkladu s vyznačenými listy a ohodnocenými orientovanými hranami

Z obrázku, na kterém je takový binární strom vyobrazen, postupujeme při šifrování, resp. dešifrování vždy od kořene stromu, dokud nepřečteme list tohoto stromu (zvýrazněny modrou barvou). Snadno bychom teď mohli přidat do tabulky další znaky přidáním dalších listů, resp. povýšením některých listů na uzly a přidávali hrany ohodnocené z abecedy {0,1} tak, že směr k levému potomku od uzlu značí hodnotu 0, k pravému pak naopak 1. Ukážeme si nyní, jak bychom přidali nový libovolný znak otevřené abecedy (např. D) do takového stromu na obrázku (Obrázek 2).



Obrázek 2 : Binární strom zachycující situaci překladové tabulky ukázkového příkladu s vyznačenými listy, přidáním nového symbolu a ohodnocenými orientovanými hranami

Tři listy A, B, C nám dávaly na výběr celkem $3 \cdot 2 = 6$ možností, kam nový symbol umístit. My jsme zvolil v našem příkladě např. pozici nalevo od listu B (nový list je pak zobrazen fialovou barvou), zatímco původní list B se povýší na uzel a na zbývající volnou levou stranu zařadíme jeho interpretaci B (viz Obrázek 3).



Obrázek 3 : Binární strom zachycující situaci překladové tabulky ukázkového příkladu s aktualizovanými listy, uzly i hranami po přidání nového symbolu

Snadno pak i z obrázku vyvodíme prefixový znak pro znak otevřené abecedy D :

```
011
```

a nový prefixový znak pro znak otevřené abecedy B:

```
010
```

O tom, na jakou pozici daný nový symbol přesně umístit, rozhodují přesně dané postupy v závislosti na typu zvoleného šifrovacího algoritmu.

Jak je z ukázkového příkladu zřejmé, pro znaky šifrové abecedy se používají symboly nad dvouprvkovou abecedou (v našem případě {0,1}), která poskytuje možnosti pro zpracování v číslicové technice. Samotná tvorba prefixových kódů je na první pohled sice jednoduchá záležitost, jejíž složitost ovšem roste s přibývajícími jedinečnými znaky otevřené abecedy, obzvláště v případech, kdy chceme zachovat pro každý znak otevřené abecedy znak šifrové abecedy minimální délky.

Algoritmy generující prefixový kód můžeme kategorizovat podle různých kritérií. Podle délky prefixového kódu na

- minimální (např. Huffmanovo kódování)
- ostatní

Podle rozsahu otevřené abecedy:

- případové (alfa, beta)
- univerzální (Shannon, Huffman)

Podle způsobů umístění klíče (stromu) pro dešifrování:

- veřejné (šifrovaný výstup v sobě obsahuje také klíč)
- soukromé (klíč je od výstupu galvanicky oddělen a dodán jinou formou)

V novodobé počítačové historii se využívají šifrovací algoritmy pro kompresi výstupních dat, musí se tedy jednat o bezeztrátové algoritmy. Při samotné kompresi, resp. dekompresi se využívá bitového přístupu. Jeden znak otevřené abecedy (s označením 0-255) zabírá velikost 1 byte, tedy 8 bitů. Myšlenka prefixových kódů při kompresi pak vychází z předpokladu, že pro zápis jednoho znaku ale není potřeba využít všech 8 bitů z jednoho byte, ale pouze jeho prefixovou část a zbývající bity využít pro další sekvenci prefixového kódu dalšího znaku otevřené abecedy. V našem ukázkovém příkladě by tedy původní sekvence CBAABB čítající celkem 6 znaků měla velikost $6 \cdot 8 = 48$ bitů, zatímco jeho zašifrovaný ekvivalent 0001110101 pouze 10 bitů[1].

6 Entropie vstupních dat

Pojem entropie vstupních dat souvisí s kompresí dat a statistickým zastoupením jednotlivých znaků otevřené abecedy v nich zastoupených. Je dána vzorcem uvedeném v [1], který zní:

$$H = -\sum_{i=1}^{n} P_i \log_2 P_i$$

kde Pi je pravděpodobnost výskytu příslušného jednoho znaku (symbolu) na vstupu, jejíž hodnota nám udává množství informace ve zprávě, v jaké míře je vhodné data komprimovat, tzn. odstraňovat redundantní informace, vhodnost komprese, resp. její efektivita, je přímo úměrná hodnotě entropie vstupu. Ve výsledku u prefixových algoritmů je pak porovnávána jednak entropie vstupních dat s entropií výstupních zašifrovaných dat, tak také entropie výstupních zašifrovaných (shodných) dat mezi jednotlivými algoritmy. Mezními hodnotami pak bývají vstupní data obsahující x-krát tentýž znak otevřené abecedy, nebo sekvenci kde jsou všechny znaky otevřené abecedy zastoupeny s rovnoměrnou pravděpodobností.
V poznatcích uvedených v [2] se pak můžeme dovědět, že entropie souboru využívají všechny kompresní algoritmy.

7 Alfa kódování

Mezi základní prefixové algoritmy patří alfa kódování. Zabývá se šifrováním dat pouze nad abecedou $M; M = \{0,1,2,3,4,5,6,7,8,9\}$. Ani samotné přidělení prefixových kódů nepatří mezi optimalizované, překladová tabulka může vypdat např. jako Tabulka 2.

[1] Při zápisu do souboru bychom jej ale museli doplnit na nejbližší celou vyšší velikost v bytech, tedy 2 byty, t.j. o celkové velikosti $2 \cdot 8 = 16$ bitů s udržováním informace o počtu korektních bitů posledního bytu, tedy o takovou informaci, která nám říká, kolik bitů je ještě šifrovaný text, a kolik už jen doplněk pro konec souboru v případě, že výsledná šifovací sekvence není beze zbytku dělitelná 8. Tato korekce ale nemá na výhodu bitového přístupu, obzvláště při práci s rozsáhlejšími daty, vliv a je zanedbána.

Znak otevřené abecedy	Prefixový kód
0	1
1	01
2	001
3	0001
4	00001
5	000001
6	0000001
7	00000001
8	000000001
9	0000000001

Tabulka 2 : Prefixy pro jednotlivé znaky vstupní abecedy pro Alfa kódování

Podle dřívější kategorizace algoritmů můžeme usoudit, že se nejedná o minimální prefixový kód a není univerzální. Navíc se nezabývá statistickým rozpoložením jednotlivých symbolů a komprese je výhodná pouze pro prvních 7 znaků (čísel). Z hlediska šifrování co se týče bezpečnosti i výhodnosti komprese by bylo vhodnější seřadit první sloupec podle počtu výskytů jednotlivých symbolů ve vstupních datech (případně náhodně) a až následně přiřazovat prefixy. Jedná se tedy o základního zástupce z třídy těchto algoritmů, který uplatnění v dnešním reálném světě příliš nenachází. Jedinou formální výhodou oproti dalším algoritmům je minimální časová složitost, která je v tomto případě dána počtem prvků znaků otevřené abecedy.

8 Beta kódování

Beta kódování je další představitel z řady jednoduchých kódů pro celá přirozená čísla. Jedná se klasickou interpretaci čísla v binární podobě bez doplňku na celý byte. Jednotlivé bity binární podoby mají každý svou váhu, která zprava doleva exponenciálně roste, přičemž na samém začátku je váha rovna hodnotě 1. Uvažujme jednoduchý příklad pro kódování čísla 25:

```
11001
```

pak bit v řetězci na poslední pozici představuje nejnižší váhu 1, bit nalevo od něj pak váhu 2, následující 4,8 a 16. Součet váh, jejichž funkční (bitová) hodnota není rovna 0 nám dává původní číslo 27.

9 Shannonovo kódování

Průkopníkem v oblasti prefixových kódů se stal v roce 1949 algoritmus známý pod názvem Shannon-Fanovo kódování, zkráceně Shannon kódování od tří autorů: Shannon, Weawer a Fano. Využivá abstraktní datové struktury binárního stromu a jeho princip je velmi podobný ilustracím a popisům v úvodu této kapitoly s tím rozdílem, že vždy pro vložení nového listu (znaku) do binárního stromu neexistuje lepší (vhodnější) pozice ve smyslu délky takto nově vzniknuvšího prefixového binárního kódu vzhledem k četnosti vkládaného znaku ve vstupním (otevřeném) souboru. Protože tedy algoritmus využívá statistického rozpoložení jednotlivých znaků a až na základě těchto informací musí vybudovat binární strom pro konstrukci všech prefixů, prochází vstupní otevřenou abecedu, resp. vstupní nešifrovaný soubor dvěma fázemi:

1) statistika abecedy (četnost znaků)
2) šifrování abecedy (kódování)

ad 1) Procházíme vstup a zapisujeme četnosti jednotlivých znaků. Takto vzniklou kolekci s nenulovými výskyty můžeme seřadit sestupně. Vytvoříme kořen stromu který bude obsahovou právě celou kolekci. Posléze rekurzivně vždy dělíme tuto kolekci na dvě části s co možná nejmenším rozdílem jejich celkovému součtu četností znaků a vložíme takto vzniklé dvě kolekce jako potomky kořenu, dokud nezískáme v jedné (či obou) takto rozdělených kolekcích právě jeden prvek (znak) který charakterizuje nový list namísto uzlu. Ukážeme na příkladě řetězce:

```
abrakadabra
```

Výskyt jednotlivých znaků v otevřené abecedě je následující

A	5
B	2
R	2
D	1
K	1

Vytvoříme kořen stromu obsahující všechny uvedené znaky. Provedeme první rozdělení intervalu, v prvním bude

```
a
```

ve druhém

```
brdk
```

Vidíme, že první interval má četnost rovnou hodnotě 5, druhý je dán součtem četností a dostáváme hodnotu 6. Rozdíl součtu četností těchto dvou intervalů je tedy 1. Zkusíme za účelem nalezení ještě menšího rozdílu přidat do prvního intervalu první znak z druhého intervalu, první kolekce pak bude vypadat:

```
ab
```

druhá pak bude ve tvaru

```
rdk
```

První interval má nyní součet četností 7 (5+2), druhý 4 (2+1+1). Rozdíl je nyní roven hodnotě 3, což je více než v případě před vložením dalšího znaku do prvního intervalu (3>1) a tento ani další znak do prvního intervalu nepatří (není možné dosáhnout rozdílu lepšího než 1) a zůstává v intervalu druhém. Dochází k definitivnímu rozdělení na dva nové potomky kořene:

```
a
```

a

```
brdk
```

Protože se pohybujeme v oblasti binárních stromů, první můžeme označit jako levý potomek, druhý pak jako pravý.
Na takto dva vzniklé uzly aplikujeme stejný postup. Protože množina prvního intervalu je jednoprvková, je označena listem a přecházíme na rozdělování pravého uzlu. Opět vložíme první znak do první kolekce a zbytek do druhé, získáme tak dvě množiny o obsahu:

```
b
```

a

```
rdk
```

Prvně jmenovaná má součet četností svých znaků rovna hodnotě 2, druhá pak 4 (2+1+1). Rozdíl je tedy 4 - 2 = 2. Zkusíme vložit do první kolekce první prvek z druhé kolekce a rozdíl (diferenci) přepočítat. První bude vypadat takto:

```
br
```

ve druhé zbyde:

```
dk
```

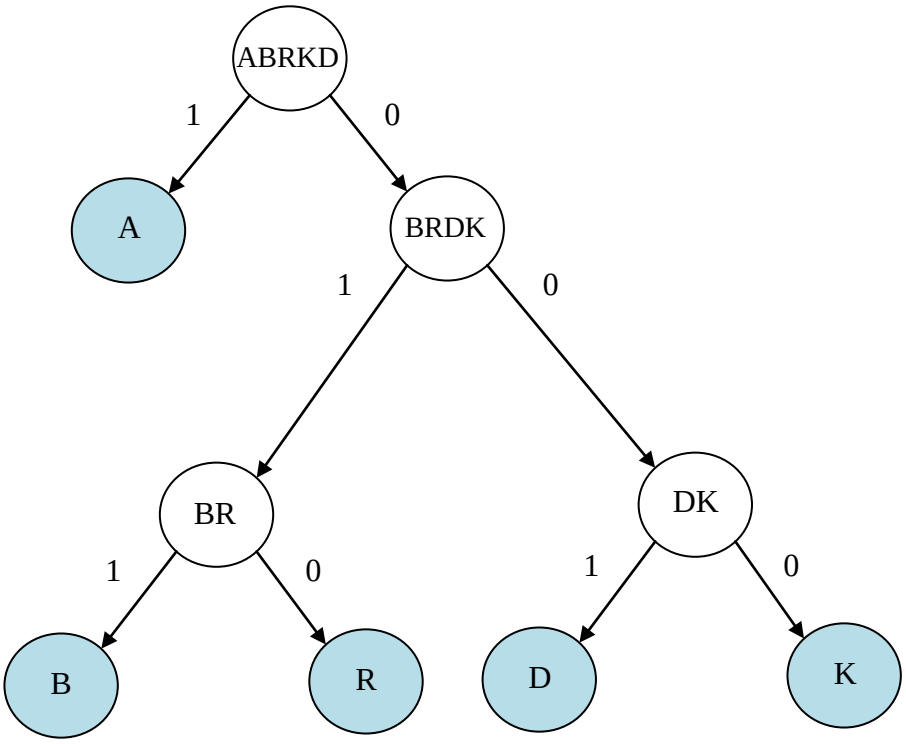
Součet četností prvního intervalu by nyní byl roven matematické operaci 2 + 2 = 4, zatímco součet v druhém intervalu by byl 1 + 1 = 2. Rozdíl součtů je nyní opět roven hodnotě 2, což není horší (větší) než před přeložením znaku z druhého do prvního intervalu a můžeme tedy prvek (znak r) v první kolekci ponechat. Je zřejmé, že dalším vložením znaku z druhé kolekce do první získáme již pouze zcela jistě horšího (většího) rozdílu, máme tedy dva nové (dvouprvkové) uzly, levý:

```
br
```

pravý:

```
dk
```

Protože obsahují právě dva prvky, je zřejmé že další rozdělování by již vždy rozdělilo interval jedním krokem na dvě části s jednoprvkovými kolekcemi a tedy listy (viz Obrázek 4).



Obrázek 4 : Příklad binárního stromu pro shannonova kódování

jednotlivých znaků (listů) v závislosti na ohodnocených hranách ve směru od kořene stromu. Např. pro znak D by prefix měl hodnotu:

```
001
```

Analogicky bychom pak postupovali pro konstrukci šifer všech zbývajících symbolů.
add 2) Vstup procházíme znova celý znak po znaku a na základě vytvořených prefixů z binárního stromu již jen přiřazujeme aktuálnímu přečtenému znaku na vstupu odpovídající prefix.

10 Huffmanovo kódování

Huffmanovo kódování je principem velmi podobné předchozímu Shannon-Fan kódování. Rozdílnost tkví v budování binárního stromu, zatímco jeho předchůdce využívá principu shora dolů, tedy budování od kořene k listům, tento algoritmus jej buduje odspodu nahoru, tedy od listů ke kořeni, díky čemuž dosahuje optimálních (nejmenších) prefixů. Taková podobnost algoritmů není náhodou, Huffman byl univerzitním žákem profesora Shannona. Dnes se kódování využívá především v kompresi dat.
Samotný algoritmus opět využívá statistického rozpoložení znaků v jedné počáteční a setříděné kolekci. Posléze vždy rekurzivně vyjme z kolekce dva symboly s nejmenší četností ve vstupním souboru a vloží je do kolekce zpět jako jeden podstrom (uzel s dvěma listy) se součtem jejich četností. Kolekci znovu setřídí a aplikuje stejný postup dokud není v

kolekci právě jeden prvek (kořen celého výsledného stromu) obsahující všechny symboly vstupní abecedy. Takovému postupu se také říká prioritní fronta stromů.

Také zde platí, že výsledné prefixy mohou být různé v závislosti na vybraných prvcích při shodě jejích četností.

11 Aritmetické kódování

Aritmetické kódování ve své teoretické definici můžeme sice zařadit do prefixových kódů, kde jejich definici neporušuje spíše díky tomu, že v ideálním případě představuje vždy právě jeden kód (číslo) pro celý vstup a nemůže se tedy stát, že by jiný prefix byl předponou jiného (druhého). V počítačové podobě ovšem narážíme na fakt, že nemáme nekonečnou soustavu (resp. nemůžeme dělit do nekonečna) a výsledných aritmetických kódů je pak vícero. Hlavním principem je přiřazení čísla z předem daného intervalu co největší sekvenci znaků ze vstupu. Opět i zde se při výpočtech využívá statistického uspořádání. Algoritmus si vysvětlíme na příkladě. Mějme na počátku interval 0-999 a řetězec

```
AABAAACAAA
```

V první fázi projdeme vstup a zjistíme četnosti jednotlivých znaků F(x), tedy F(A) = 7, F(B) = 2, F(C) = 1. Kolekci nemusíme třídit. Přiřadíme každému znaku v kolekci interval, jehož rozsah je závislý na četnosti znaku tak, abychom využili celý rozsah 0-999. V našem případě pro znaky A B a C dosáhneme rozdělení počátečního intervalu do podoby tabulky (viz Tabulka 3).

Znakový interval	Poč. interval
A	0-699
B	700-899
C	900-999

Tabulka 3 : Příklad aritmetického kódování - počáteční stav

V druhé fázi již čteme vstup znak po znaku. V našem případě čteme první znak A a z tabulky najdeme příslušný interval a rozsah. Vidíme, že se jedná o první řádek (Znakový interval A) s rozsahem 0-699. Takový interval se stane novým hlavním intervalem, který musíme znovu přerozdělit vzhledem k dalším znakům, dostaneme tedy tabulku s dalším sloupcem, viz Tabulka 4.

Znakový interval	Poč. interval	Interval 1
A	0-699	0-489
B	700-899	490-639
C	900-999	640-699

Tabulka 4 : Příklad aritmetického kódování - přečtení prvního znaku

Vezmeme další, resp. druhý znak ze vstupu. Opět se jedná o znak A. Podíváme se do tabulky rozsahů intervalů na řádek, který odpovídá znakovému intervalu A a poslednímu sloupci. Vidíme že se jedná o rozsah 0-489. Tento rozsah tedy opět přerozdělíme vůči četnostem všech znaků, dostaneme pak výsledek viděný v tabulce (viz Tabulka 5).

Znakový interval	Poč. interval	Interval 1	Interval 2
A	0-699	0-489	0-342
B	700-899	490-639	343-440
C	900-999	640-699	441-489

Tabulka 5 : Příklad aritmetického kódování - přečtení druhého znaku

Třetí přečtený znak na vstupu je nyní B. Vezmeme hodnotu intervalu jako poslední sloupec a druhý řádek z tabulky, kterou je 343 - 440 a tento opět přerozdělíme (viz Tabulka 6).

Znakový interval	Poč. interval	Interval 1	Interval 2	Interval 3
A	0-699	0-489	0-342	343-411
B	700-899	490-639	343-440	412-431
C	900-999	640-699	441-489	432-440

Tabulka 6 : Příklad aritmetického kódování - přečtení třetího znaku

Analogicky bychom postupovali pro všechny ostatní znaky. Pak můžou nastat dva případy:
1) konec vstupní abecedy
Celému vstupu pak odpovídá číslo jako dolní mez z intervalu posledního znaku posledního sloupce. V našem případě, pokud by třetí znak (B) byl posledním, pak by celému vstupu odpovídala hodnota 412.
2) konec rozdělování intervalu
V případě, že již nemůžeme rozdělit interval (měl by nulovou nebo zápornou délku) pak si musíme uložit číslo právě přečtené sekvenci jako by se jednalo o konec vstupu, tuto hodnotu a sekvenci si uložit do paměti, a pro další znak již přerozdělovat znovu celý počáteční (inicializační) interval 0-999.

12 Závěr

Vysvětlili a naimplementovali jsme si vybrané partie z prefixových algoritmů a doplnili jsme je o některé další segmenty, jako např. aritmetické kódování nebo srovnání jednoho z průkopnických algoritmů s algoritmem určeným pro šifrování obrázků.
Těmto experimentům se podobněji a prakticky zabývá příloha "A" s referencí na konci dokumentu. Na základě generovaných výstupů algoritmů generujících prefixové kódy pro libovolný vstup (převážně ale textového typu) dochází k jeho zašifrování, při použití bitového přístupu k souborům pak, tak jako v naši implementaci, dochází ke kompresi (zmenšení) souboru, tzn. je využito volných bitů u znaků, které je v danou chvíli nevyužívají a odstranění redundancí. V implementační části bylo využito několika principů z [2], jakými jsou např. techniky pro čistý (srozumitelný) kód, přídavné komentáře pouze pro segmenty s veřejným modifikátorem apod. Rozšířením této práce by mohlo být např. popis dalších méně známých prefixových algoritmů a jejich vzájemné srovnání, či implementace v dalších programovacích jazycích (C++, Java) za účelem dalšího srovnání, tentokráte výkonnosti platforem na těchto algoritmech nad totožnými vstupními daty a použitém algoritmu.

13 Seznam obrázků

14 Seznam tabulek

15 Literatura

[1] **Ph.D, Ing. Platoš Jan.** Komprese dat. *Předmět komprese dat na VŠB-TU OSTRAVA.* [Online] 2011. [Citace: 5. duben 2011.] http://homel.vsb.cz/~pla06.
[2] **C., Martin Robert.** *Čistý kód.* Brno : Computer Press, a.s., 2009. ISBN 978-80-251-2285-3.
[3] **Arnošt, Večerka.** *Komprese dat.* Olomouc : autor neznámý, 2008.

16 Přílohy

A Výsledky experimentů předmětu Komprese dat, Téma: Cvičení 7 - Komprese obrazových souborů :

```
Project doc068/Attachments/A.pdf
```