Adaptivní Huffmanovo kódování

Základní idea Huffmanova kódování spočívá v použití prefixových kódů různých délek, přičemž nejčastěji vyskytující se znaky se zakódují nejkratším prefixem. Kompresní algoritmus se typicky používá dvojím způsobem, buďto staticky, nebo adaptivně. Statická metoda je dvoufázová, nejprve se udělá statistika a podle ní se vypočtou prefixové kódy. V druhé fázi se provede samotné zakódování. Nyní bych chtěl čtenáře poprosit, aby se dokonale seznámil se statickou metodou komprese, která byla představena na přednášce. Čtení dalšího textu by bez její znalosti nemělo smysl. Problematika je popsána rovněž v knize [SICP].

Naproti tomu adaptivní varianta algoritmu pracuje pouze jednoprůchodově. Na počátku komprese se uvažuje, že všechny znaky abecedy mají *četnost výskytu rovnu nule.* Během komprese se aktualizuje statistika o výskytu znaků a podle ní se přidělují znakům prefixy, kterými jsou dále zakódovány. Ukažme si motivační příklad na vysvětlenou. Dejme tomu, že zakódujeme adaptivní metodou řetězce znaků „`X`“, „`XX`“, „`XY`“ a „`XYXYY`“. Viz následující schéma.

vstup	*výstup – binárně*
`X`	`001`
`XX`	`001 1`
`XY`	`001 001`
`XYXYY`	`001 001 01 01 1`

Předpokládejme, že pracujeme pouze s pětadvacetiprvkovou abecedou „`A`“ až „`Z`“. Na počátku je všem znakům přidělena četnost nula, tedy znaky „`X`“ a „`Y`“ jsou kódovány posloupnostmi „`001`“ a „`01`“. V prvním případě tak tomu skutečně bylo. V druhém řádku tabulky je první výskyt znaku „`X`“ zakódován posloupností „`001`“, druhý znak je ale již zakódován jednoprvkovou posloupností „`1`“, protože po výskytu prvního znaku „`X`“ byly přepočteny prefixy kódů a znak „`X`“ měl největší četnost (má jeden výskyt, ostatní znaky byly zatím bez výskytu), a proto mu byl přidělen nejkratší prefixový kód „`1`“. Z dalšího řádku tabulky je vidět, že po přepočtení prefixů může dojít k tomu, že dva různé znaky jsou těsně po sobě zakódovány stejnými kódy. Konečně z posledního řádku tabulky je vidět, jak se jednotlivým znakům mění prefixové kódy v závislosti na četnosti jejich výskytů. Znak „`Y`“ byl nakonec zakódován jednoznakovým kódem „`1`“, protože na páté posici ve vstupním řetězci měl četnost tři, kdežto znak „`X`“ pouze dva. Srovnej se začátkem kódu.

Implementace. Nyní krátké poznámky k implementaci. Algoritmus zakódování je tedy stejný jako u statické metody, pouze po každém zakódovaném znaku budeme znovu generovat prefixové kódy. Stejně tak při dekompresi. Výstupem obou algoritmů jsou tedy pouze vlastní data, odpadá potřeba uchovávat informace o kódech.

Při implementaci uvažujme pro jednoduchost pouze řetězce sestavené ze znaků „`A`“ až „`Z`“ bez mezer. Při práci se znaky je dobré dbát na fakt, že znaky se musejí porovnávat predikátem `equal?`, nikoliv predikátem rovnosti „`=`“. Při práci se znaky budete možná potřebovat procedury pro konverzi znaku na jeho ASCII hodnotu a obráceně, k tomu slouží procedury `char->integer` a `integer->char`. Při práci se znaky se používají vnitřní datové typy *znak* a *řetězec.* S výhodou lze využít i existující procedury jazyka Scheme `string->list` a `list->string`, které transformují řetězec na seznam znaků a obráceně.

Pro konverzi seznamu jedniček a nul na řetězec znaků a obráceně použijte procedury z knihovny `bitstream.scm`. Pro účely ladění ale doporučuji nejprve pracovat pouze se seznamy jedniček a nul, jinak se v tom utopíte.

Reference

[SICP] Abelson, H. – Sussman, G. J. *Structure and Interpretation of Computer Programs.* The MIT Press, Cambridge, Massachusetts, 1985.

[DS99] Dvorský, Jiří – Snášel, Václav. *Algoritmická matematika I.* VUP, Olomouc, 1999. Strany 206–208.