Jednoduché kódy a Samoopravné kódy

Peter Csiba, petherz@gmail.com, http://www.csip.sk?p=652

05.06.2011

Contents

1 Úvod

Táto prednáška je zameraná populárne. Nebudeme nič dokazovať poriadne, ide nám o to zaujať tematikou. Od publika sa vyžaduje iba to, aby sa často pýtali.

Najprv sa budeme zaoberať základnými bezstratovými kódovacími algoritmami ako Huffman alebo Fannov kód. Potom sa ponoríme do hlbín samoopravných kódov a budeme sa pýtať na jeho a vlastnosti a či sa nám to vôbec oplatí robiť. Nakoniec sa posnažíme demonštrovať pokročilejší samoopravný kód opravujúci 2, resp. 3 chyby.

1.1 Používané pojmy

- Znak - nejaký prvok, väčšinou sú označované $a, b, c, \cdots$ alebo $0, 1$.
- Abeceda - konečná množina znakov.
- Slovo - konečné zreťazenie znakov abecedy (postupnosť). Štandardne ho zapisujeme *abbababa*.
- Jazyk - množina slov (môže byť aj nekonečná).
- Prefix - slovo u je prefixom slova v ak sa v dá písať ako $v = u.v_{n_0}v_{n_1}\cdots v_{n_k}$
- Váha slova - počet nenulových znakov (väčšinou sa používa pri binárnych slovách).
- Bijekcia - jednoznačné zobrazenie medzi dvoma množinami (v tomto texte väčšinou jazykmi).
- Zdroj - zariadenie, ktoré nám dáva znaky
- Kanál - rozhranie, cez ktoré posielame správy.
- Kódovanie - (bijekcia) transformácia zdrojovej abecedy do kódových slov.
- Dekódovanie - inverzná transformácia ku kódovaniu.
- Entropia zdroja - akú veľkú informáciu nesú znaky generované zdrojom. Napríklad ak generuje samé 0, tak je entropia zdroja nulová. Ak genruje všetky znaky rovnomerne, tak je maximálna a rovná sa $lg(m)$ kde m je počet znakov zdrojovej abecedy a lg je logaritmus so základom 2.

2 Bezstratová kompresia

Použiť bezstratovú kompresiu znamená transformvať vstup V tak, aby sme po jeho spätnom transformovaní dostali zase V. Napríklad všetky archivačné algoritmy (ZIP, Rar, Tar, $\cdots$) a *png* sú bezstratové. Na druhej strane, ak potrebujeme výrazne redukovať veľkosť výstupného súboru a nevadí nám potenciolnálne zníženie kvality, tak pužívame stratové kompresné algoritmy ako Jpeg, Avi a podobne.

2.1 Rozdelitelný kód

Majme nejaké kódové slová. Potom za *rozdelitelný kód* považujeme takú množinu slov, keď z každého slova nad kódovou abecedou vieme jednoznačne určiť o aké kódové slová sa jednalo (alebo zistíme, že dané slovo nie je kódové).

Príklad

- $\{01, 001, 0001\}$ tvorí rozdelitelný kód (vždy sa pozrieme, koľko 0 je pred 1.
- $\{01, 101, 1011\}$ netvorí rozdelitelný kód, lebo slovo 101101 vieme rozdeliť na viaceré, napr. $101, 101$ a $1011, 01$.
- $\{01, 10, 101\}$ to, či je tento kód rozdelitelnýc nechávame na usilovného čitateľa.

2.2 Nerovnomerný kód a Pravdepodobnostný zdroj

Za nerovnomerný kód považujeme taký, v ktorom sú dve slová rôznej dĺžky. Toto kódovanie sa používa často kvôli rôznej pravdepodobnosti znakov zdrojovej abecedy. Na generovanie takých vstupných slov používame pravdepodobnostný zdroj. Na demonštráciu uvedieme príklad s početnosťou niektorých znakov v nejakom texte:

- a - 0.25
- b - 0.10
- c - 0.12
- d - 0.13
- e - 0.20
- f - 0.05
- g - 0.07
- h - 0.08

2.3 Prefixové kódy

Za špeciálny typ rozdeliteľných nerovnomerných kódov považuje prefixové kódy. Sú založené na tom, že žiadne slovo z kódovej abecedy nie je prefixom iného slova z kódovej abecedy.

Príklad

- $\{01, 10, 101\}$ nie je prefixovým kódom, lebo 01 je prefixom 101.
- $\{01, 10, 00\}$ je prefixovým kódom, lebo žiadne slovo nie je prefixom iného.

Všimneme si, že pre prefixové kódy vieme zostrojiť binárne ohodnotené zakorenené stromy. Čítaním znakov počas cesty z koreňa po list získame jednotlivé kódové slová.

2.4 Fannov kód

Prefixový kód pre pravdepodobnostný zdroj. Tvorí sa iteratívne spôsobom rozdeľuj a panuj. Najprv si zoradíme zdrojové znaky podľa pravdepodobnosti ich výskytu. Potom ich rozdelíme na dve súvislé časti tak, aby rozdiel súčtu pravdepodobností v tých dvoch častiach bol čo najmenší. Horné slová sa budú začínať na 0 a dolné slová na 1. Takto pokračujeme kým je čo rozdeľovať. –Demonštrácia na tabuli.

2.5 Huffmanov kód

Podobne ako Fannov, je prefixový a slúži na kódovanie pravdepodobnostného zdroja. Takisto je iteratívny a v prvej fáze si zoradíme znaky podľa pravdepodobnosti. Postupne tvoríme binárny strom. Zoberieme dva najnepravdepodobnejšie znaky a zlúčime ich do nového. V strome sa tým vytvorí otec - nový znak - s jeho synmi - zlučované znaky. Nový znak bude mať pravdepodobnosť súčtu predošlých dvoch. –Demonštrácia na tabuli.

Dá sa ukázať, že Huffmanov kód je optimálny. Je zaujímavé, že pri rovnosti pravdepodobností (aj tých zlúčených) je jedno, ktorú vyberieme na ďalšie zlučovanie.

3 Chyby detekujúce a samoopravné kódy

Napriek tomu, že vieme generovať optimálne kódy, je možné, že v kanály nastane šum a niektoré bity kódových slov sa prevráti. Často je horšie dostať zlú spávu (napr. namiesto neútočte - útočte) ako nedostať žiadnu správu. Preto sa do správ často pridáva informácia navyše (tým sa síce zníži entropia, ale zvýši istota).

V ďalšom sa už nebudeme zaoberať kódovými slovami, ale iba časťami vysielanej správy. Zreťazené kódové slová rozdelíme na rovnako veľké časti a ich budeme ďalej manipulovať. Tieto časti budeme nazývať slová.

Napriek našej úpornej snahe nevieme vylúčiť možnosť nedetekovanej chyby (okrem prípadu, keď posielame stále to isté). Vieme ju ale redukovať na pravdepodobnosť o ktorej ani nemá zmysel uvažovať, či nastane.

Šum v kanály Matematicky interpretujeme šum ako zmenu bitu s pravdepodobnosťou p. V rôznych médiách prichádza k rôzne silným (v zmysle veľkosti pravdepodobnosti) šumom. Pri bezdrátovej je to okolo 10^{-4}, pri medených likách okolo 10^{-7} a pri optických okolo 10^{-11}. Uvedomme si, že pravdepodobnosť viacerych chýb exponenciálne klesá. V skutočnom svete občas prichádza k *burst error* - veľa chýb za sebou (rušička, vuvuzela, $\cdots$).

Poznámka: Kvalitný generovaný šum je celkom celným artiklom.

3.1 Základné

Slovo vieme rozdeliť na *informačné znaky* - to sú tie ktoré nesú informáciu a *kontrolné znaky* - tie ktoré nesú informáciu o informačných znakoch.

3.1.1 Kontrolné súčty

Anglicky nazývané checksumy. Myšlienka je taká, že za každé slovo pripíšeme niekoľko bitov tak, aby počet jednotiek v slove bol deliteľný nejakým číslom. Napríklad ak chceme, aby bol súčet deliteľný dvoma, tak nám stačí pripísať jeden bit. Takým spôsobom vieme detekovať nepárny počet chýb. Ak zoberieme v úvahu, že viacej chýb je nepravdepodobných, tak nám to stačí. –Demonštrácia na tabulu.

3.1.2 Obdĺžnikový prístup

Slovo zapíšeme do obdĺžnika (ideálne štvorca). Za každý riadok a za každý stĺpec dorátame checksum. A ešte vypočítame checksum checksumov. Týmto spôsobom vieme opravovať jednu chybu a detekovať chyby váhy nedelitelnej štvorkou a opravovať jednu chybu. Rozmyslite si, čo sa stane ak chyba vznikla v checksume, respektíve checksume checksumov. –Demonštrácia na tabulu.

3.1.3 Hammingov kód

Obdĺžnikový prístup vieme zefektívniť a rozšíriť na viacero rozmerov. Hammingov kód vieme zostrojiť pre $2^m - m - 1$ informačných znakov a m kódových znakov. Vieme aj dokázať, že sú v istom zmysle optimálne čo sa týka kódov opravujúcich jednu chybu. Myšlienku si demonštrujeme na štvorrozmernom prípade. – Demonštrácia kocky a kontrolných súčtov pre jednotky, dvojky a štvorky v binárnych zápisoch (nie je to to isté ako Hamming.

Hammingov(15, 11) kód Vytvoríme si checksumy pre jednotky na každom zo štyroch miest binárneho zápisu pozícií. Tieto checksumy si uložíme na pozíciách 1,2,4 a 8. Tým pádom, ak nebudú sedieť checksumy na niektorých pozíciách, tak nám budú udávať binárny zápis miesta, kde prišlo k chybe.

3.2 Mierne pokročilé

Zoberme si vektorový priestor všetkých binárnych slov dĺžky n. Definujeme si na ňom vzdialenostnú funkciu (metriku) takú, že $dist(u, v) = weight(u - v)$. Aby sme boli schopný opravovať t chýb, potrebujeme, aby vzdialenosť každých dvoch slov bola aspoň $2t+1$. Inak povedané, v sférickom (guľovom) okolí slova polomeru $2t$ nemôže byť žiadne iné kódové slovo.

V ďalšom texte už v plnej miere upustíme od zdôvodnenia, prečo vektory sú vzdialené aspoň toľko. Vyžaduje to znalosti z vysokoškolskej algebry.

Označenia

- $i(x)$ - informačný polynóm
- $g(x)$ - generujúci polynóm
- $u(x)$ - $u(x) = i(x) * g(x)$ - kódový polynóm
- $e(x)$ - chybový polynóm
- $v(x)$ - vstupný polynóm na dekódovanie

3.2.1 Lineárne kódy

Požadujeme, že ak u, v sú kódové vektory, tak aj $u+v$ sú kódové vektory (násobenie skalárom je v binárnom prípade ozaj triviálne). –Demonštrácia na tabuli ako taký priestor vyzerá a znázornenie vzdialeností medzi nim.

Vektory budú vlastne inak reprezentované polynómy.

Kódovanie Budeme kódovať informačné vektory - pozostávajúce z informačných znakov. Kódovať budeme tak, že vynásobíme informačný vektor generujúcim vektorom ako keby to boli polynómy (to znamená, že $1 + 1 = 0$ bezo zvyšku).

Dekódovanie Pri kódovaní sme násobili polynómy, teraz ich budeme deliť.

3.2.2 Inštancia BCH a demonštrácia kódovania a dekódovania

$g(x) = x^3+x+1$ – jedna chyba. $g(x) = x^8+x^7+x^6+x^4+1$ – dve chyby. $g(x) = x^10+x^8+x^5+x^4+x^2+x+1$.

4 Záver

Okrem Hammingovho a Golayovho kódu nepoznáme iné optimálne samoopravné kódy. Je pomerne ťažké rozdeliť daný vektorový priestor na disjuktné sféry ktoré ho celé pokrývajú.

Dúfam, že táto prednáška oslovila čitateľov a prezentovala teóriu kódovania v dobrom svetle. Akýkoľvek feedback posielajte prosím vás emailom (adresa na začiatku), alebo do komentárov na mojej homepage (adresa na začiatku).