Föreläsning 8

MA1508

Komprimering

När Alice och Bob kommunicerar har de begränsad bandbredd. Även med modernt bredband så finns det alltid begränsningar i hur många bitar de kan skicka per sekund, och det finns alltid önskemål om att göra det ännu snabbare. Likaså, på hårddiskar och andra lagringsmedier är minnesutrymmet ibland ett problem och vi skulle önska vi kunde få använda mindre plats för att spara vår data utan att bli av med någon information.

För att att försöka lösa detta problem kommer vi använda *komprimering*, och mer specifikt förlustfri komprimering. Det är vanligt när man överför t ex ljudfiler att man bestämmer sig för att spara lite data på bekostnad av att göra ljudet lite sämre. Det är ofta en bra lösning och det finns intressant teori för det, men det är inte den typen av komprimering vi kommer ägna oss åt, utan vi vill kunna återskapa det ursprungliga meddelandet/datan exakt från den komprimerade datan.

Det är värt att nämna en fundamental begränsning för det. Låt oss tänka på fallet då vi komprimerar filer i en dator. Om vi har en komprimeringsalgoritm som tar en fil och ger oss en ny fil så kan vi se det som att vi har en funktion från binära strängar till binära strängar. Låt oss kalla funktionen för f. Att komprimeringen ska vara förlustfri betyder att f är en injektiv funktion. Att f komprimerar betyder att åtminstone ibland är $f(u)$ kortare än u. Men det kan inte hända utan att $f(v)$ ibland är längre än v. (Låt nämligen M vara mängden av alla strängar med längd högst n. Om f avbildar M till sig själv så måste f vara bijektiv som funktion från M till M, enligt lådprincipen. Alltså kan ingen sträng utanför M avbildas till M, om f ska fortsätta vara injektiv.)

Alla förlustfria komprimeringsmetoder måste alltså i teorin riskera att istället expandera vissa filer.

Motiverande exempel

Låt oss diskutera ett exempel modifierat från boken som de använder för att motivera den första komprimeringsmetoden vi kommer prata om. Ni vet att det kommer bli Huffman-koder, men låt oss ändå fundera på lite alternativ.

Alice, Bob och Carolina är tre vänner du ofta ringer. Du ringer Bob 50% av tiden, Alice och Carolina 25% av tiden. Du vill anteckna vem du ringt så du kan komma ihåg det i efterhand. Du tänker använda en binär kod för att föra dina anteckningar.

Du ska ange 3 alternativ. Om alla ska ha lika långa koder behöver alla ha 2 bitar i sin kod. Då behöver du inte tänka särskilt på hur du tilldelar koderna, bara du inte ger två personer samma nummer. Känns som du behöver anteckna onödigt många siffror i genomsnitt.

Boken föreslår istället att Alice får koden 1, Bob 0 och Carolina 01. Men det har det uppenbara problemet att står det 01 vet du inte om du ringde Bob och sen Alice, eller istället ringde Carolina en gång.

En optimal kod är givetvis att tilldela Bob koden 0, Alice koden 10 och Carolina koden 11.

Det gör koden *unikt avkodningsbar*, vilket innebär att om du får se en följd av ettor och nollor som koden genererat så finns det bara ett sätt att dela upp den i kodord.

Om du t ex skriver ner 10110010 så måste det betyda Alice, Carolina, Bob, Bob, Alice.

Här är en annan intressant kod: Bob har koden 0, Alice har koden 01 och Carolina har koden 11. Nu om du ser en 0:a så vet du inte om det i sig representerar Bob eller om det är början på koden för Alice. Men jag påstår koden ändå är unikt avkodningsbar. Om du t ex skrivit ned 0010111 så måste det faktiskt komma från sekvensen Bob, Alice, Alice, Carolina. Den genomsnittliga kodlängden är samma som förra koden Så denna koden verkar lika optimal som förra på sätt och vis.

Koden med orden 0, 10, 11 är en *prefixfri kod*. Ett prefix är en inledning på ett ord, och kravet på en prefixfri kod är att inget kodord är ett prefix till ett annat kodord. Det innebär att så fort vi känner igen ett kodord kan vi tolka det. Det finns andra koder än prefixkoder som är unikt avkodningsbara, som vi såg ovan, men prefixkoder är enkla att jobba med.

Huffmankoderna som vi kommer skapa är prefixkoder. Vi ska senare diskutera om vi förlorar något på det.

Koder

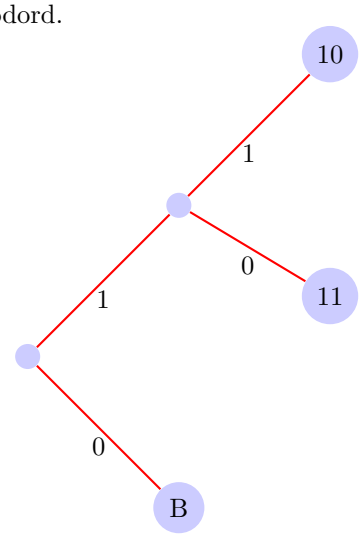
När vi pratar om komprimering kommer vi använda ordet koder i annan betydelse än vi gjorde i förra kapitlet. Nu är en kod en funktion som till element i en ändlig mängd M tilldelar binära strängar. De strängarna som tilldelas kommer kallas kodord. Strängarna behöver inte vara lika långa för alla element i M.

Binära träd

Ett *binärt träd* är en graf som är ett träd i den mening ni lärde er i diskret matematik med det ytterligare kravet att vi pekar ut en av noderna som en

rot, och vi kräver att varje hörn utom löven har exakt två barn, dvs två kanter som lämnar det i riktning bort från roten. Hörnen i grafen delar boken in i noder och löv, där noderna är alla kanter utom löven.

Vi kan representera prefixfria koder med binära träd. Här är t ex en representation av den koden 0, 10, 11, utom informationen vem som tilldelas vilket kodord.



Vill vi tala om vem som får vilken kod skulle vi kunna skriva det i löven också, eller istället för själva kodordet. För från sekvensen av grenar kan vi avläsa vad kodordet är.

En prefixfri kod kan lämna vissa löv tomma.

Definition. *Djupet* hos ett löv är hur många steg man tar när man vandrar kortaste vägen från roten till lövet.

Sats. *Om ett binärt träd har n löv, N stycken noder och lövens djup är $d_1, d_2, \ldots, d_n$ så gäller att*

$$n = 1 + N$$

och

$$\sum_{i=1}^{n} 2^{-d_i} = 1.$$

Ett skiss på ett bevis är att notera att det är sant om $N = 1$, och att om det är sant för ett träd så förblir det sant om vi gör om ett av löven till en nod med två löv som barn.

Kopplingen mellan träd och koder låter oss formulera nästa sats.

Sats (Krafts olikhet)**.** *Det finns en prefixfri kod med r stycken kodord vars längder är $\ell_1, \ell_2, \ldots, \ell_r$ om och endast om*

$$\sum_{i=1}^{r} 2^{-\ell_i} \leq 1.$$

Exempel 1. Finns det en prefixfri kod med 3 kodord av längden 2 och 3 kodord av längden 3?

Svaret är nej, för

$$2^{-2} + 2^{-2} + 2^{-2} + 2^{-3} + 2^{-3} + 2^{-3} = \frac{9}{8},$$

och $\frac{9}{8}$ är större än 1. Men däremot finns det en prefixfri kod med 3 kodord av längden 2 och 2 kodord av längden 3. △

Beviset för Krafts olikhet utnyttjar kopplingen till binära träd.