Föreläsning 4

MA1508

Felrättande och felupptäckande koder

Återigen är vi i situationen att Alice och Bob kommunicerar över en kanal som med sannolikheten p flippar en bit.

Skickar Alice ett meddelande som är n bitar långt är sannolikheten för att det blir exakt ett fel vid överföringen lika med

$$np(1-p)^{n-1}.$$

Sannolikheten för exakt k fel är

$$\binom{n}{k}p^k(1-p)^{n-k}.$$

Om p är litet och n inte allt för stort så kommer sannolikheten för 1 fel vara mycket större än sannolikheten för mer än 1 fel. Den första koden vi nämner kommer låta oss upptäcka att något är fel om det sker exakt 1 fel. (Vi tänker oss att om mottagaren Bob åtminstone vet om att något är fel så kan han begära att Alice sänder meddelandet igen.)

Det gör vi genom att till meddelandet om n bitar lägger vi till en *paritetsbit*, dvs en bit som garanterar att summan av alla bitarna är 0, modulo 2.

Exempel 1. Anta Alice skickar meddelandet som är 6 bitar långa, varav den sista är paritetsbiten. Om Alices ursprungliga meddelande är 01100 så är summan av alla bitarna redan 0, för $0+1+1+0+0=0$. Så kodordet hon skickar blir 011000.

Om meddelandet hon egentligen vill överföra är 10101 så blir kodordet hon överför 101011.

△

När Bob tar emot ett överfört meddelande kollar han om bitarna summerar till 0. Gör de inte det begär han att Alice sänder om meddelandet. Bob kommer upptäcka att något blivit fel så länge det sker ett udda antal fel.

Sannolikheten för att ett udda antal fel sker är

$$\binom{n}{1}p(1-p)^{n-1}+\binom{n}{3}p^3(1-p)^{n-3}+\ldots+\binom{n}{n'}p^{n'}(1-p)^{n-n'},$$

där jag har använt n' för att betyda n om n är udda, och $n-1$ om n är jämnt.

Boken har en elegant härledning (som vi absolut inte kommer testa er på) som visar att den summan är lika med

$$\frac{1-(1-2p)^n}{2}.$$

Sannolikheten för att antalet fel är jämnt och större än 1 visar boken är lika med

$$\frac{1+(1-2p)^n}{2}-(1-p)^n.$$

Dessa formler använder boken på ett elegant sätt, men låt oss göra det lite enklare för oss. Sannolikheten för exakt 1 fel är $np(1-p)^{n-1}$. Sannolikheten för exakt 2 fel är $\frac{n}{2}p^2(1-p)^{n-2} = \frac{n(n-1)}{2}p^2(1-p)^{n-2}$. Om

$$np(1-p)^{n-1} > \frac{n(n-1)}{2}p^2(1-p)^{n-2}$$

är sannolikheten för 1 fel större än för 2 fel. Om vänstersidan är mycket större än högersidan kan vi nästan helt strunta i att oroa oss för 2 fel, i jämförelse med ett fel. Det kommer sedan också gälla att 3 fel är troligare än 4 fel, 5 fel är troligare än 6 fel, osv. Sammantaget blir alltså fallet att ett udda antal fel sker mycket vanligare än fallet att ett jämnt antal fel sker.

Exempel 2. Om $n = 6$ och $p = 0.001$ så är sannolikheten för exakt 1 fel lika med

$$6 \cdot 0.001 \cdot 0.999^5 = 0.0057.$$

Sannolikheten för exakt två fel är

$$\frac{6 \cdot 5}{2} \cdot 0.001^2 \cdot 0.999^4 \approx 0.0000149.$$

Sannolikheten är enormt mycket större att ett fel inträffar än 2.

△

Om Alice använder en paritetsbit när hon skickar meddelande så är det bara $n-1$ utav n bitar som verkligen innehåller någon information. Att jobba med långa block kan alltså verka frestande, men då ökar också risken att fel inträffar som gör att blocket måste återsändas, vilket så klart skadar effektiviteten. Att balansera de två faktorerna får man göra i varje enskilt fall.

Paritetsbitar användes t ex ofta i ASCII-koden, och dess varianter, som användes för att lagra text i datorer, men ursprungligen för att kommunicera mellan datorer och skrivare.

Om koder och kodord

Det kan vara värt att prata om vad vi menar med en *kod*. Detta är ett ord som används i många olika sammanhang, men när vi pratar om felrättade koder

kommer vi anta att Alice och Bob skickar block om n bitar i taget. Inte alla block behöver vara tillåtna för Alice att skicka, för vi vill ju kunna upptäcka och rätta fel om sådana uppstår vid överföringen. Om alla block är tillåtna kan Bob inte upptäcka något är fel. Istället vill man att Alice bara skickar något av vissa block, och förhoppningen är att om det blir fel i överföringen så kan Bob gissa vilket det rätta blocket var, eller åtminstone upptäcka att något har gått snett. Listan på de tillåtna blocken kallas för kodord.

T ex vid den enkla paritetskoden som vi diskuterade ovan så var de tillåtna kodorden de där summan av alla bitarna summerade till 0. Ofta kommer vi inte diskutera vad de olika kodorden betyder. Det måste Alice och Bob också vara överens om, men det är inte ett problem inom teorin för felrättande koder.

Målet med kodningsteori är till stor del att designa koder med många kodord som ändå gör det möjligt att avgöra vilket kodord som verkligen skickades, om det smyger sig in några fel i meddelandet.

Att upptäcka flera fel i rad

Anta vi inte oroar oss över fel i bara en enstaka bit utan också att två intilliggande bitar båda flippas. Så att 1101 kan bli 1011 om de två mittersta bitarna flippas. Men vi bestämmer oss att inte bekymra oss över möjligheten över att två bitar på längre avstånd båda flippas. Det kan ibland vara en mer realistisk modell för fel, t ex om fel beror på någon sorts störning som kan vara lika länge som tiden det tar att skicka två bitar. Däremot skulle att det blir fel i två bitar långt från varandra motsvara att två helt oberoende fel inträffar, vilket vi kan se som osannolikt Vi nöjer oss fortfarande med att upptäcka fel, så att Bob kan begära återsändning.

Vi bestämmer oss för att kodorden nu ska ha egenskapen att bitarna på de udda platserna och bitarna på de jämna platserna var för sig ska summera till 0, modulo 2. Om blocklängden n är 6 så är t ex 001111 ett kodord, men 111001 är inte det. Tar Bob emot det senare begär han återsändning.

Med denna kod kommer Bob upptäcka det sker fel, så länge felen innebär att endast en bit, eller endast två närliggande bitar påverkas. Sådana fel kan nämligen aldrig ändra två bitar som både står på en udda plats eller två bitar som båda står på en jämn plats.

Boken pratar om bursts av fel. På svenska kan jag kalla det för *skurar* av fel. En *felskur* av längd L innebär alla bitar som flippas befinner sig inom avstånd L från varandra. Så om ett kodord 110011 görs om till 100111 genom att andra och fjärde biten flippas så är det en felskur av längd 3. Notera att det inte krävs att den tredje biten också flippas, men det hade fortfarande varit en felskur av längd 3 om den också flippats. Vi kan skapa en kod som upptäcker felskurar av längd L på liknande sätt som vi gjorde för felskurar av längd 2. Vi kräver att bitarna på platserna $1, L+1, 2L+1$ osv ska summera till 0. Likaså ska bitarna på plats $2, L+2, 2L+2$ osv.

Mer matematiskt uttryckt kräver vi att för $i \in \{0, 1, \ldots, L-1\}$ ska det gälla att om vi summerar alla bitarna vars plats är ekvivalent med i modulo L

så ska svara bli 0.

Exempel 3. Om vi använder blocklängden 9 och gör som ovan för att skydda mot felskurar av längd 3 så blir 110001111 ett tillåtet kodord.

Jag kan konstruera ett kodord genom att välja de första 6 bitarna godtyckligt och sedan räkna ut vad de sista 3 bitarna ska vara. Det säger något om hur effektiv koden är på att överföra information, vilket vi kommer återkomma till.

△

Att förebygga mänskliga fel

Anta att vi vill att människor ska knappa in något sorts nummer, t ex sitt personnummer eller kreditkortsnummer. Det kan uppstå fel då också, och tankesättet med felupptäckande koder vi använt förut kan kanske vara relevant vara att upptäcka att något är fel. Samtidigt finns det två viktiga skillnader mot situationen vi pratat om förut. Dels så är numren förmodligen decimaltal, så det är 10 siffror som används, inte 2. Dels så är det troliga felen lite annorlunda när det är en människa som människa kan begå dem, än vad som uppstår på grund av elektriska störningar. Ett fel som man velat designa koder för att upptäcka är när man bytt plats på två siffror. Det är inte så troligt att en fysisk kanal bitar plats på två bitar, men kan lätt hända när en människa skriver.

För den första koden boken beskriver så använder den bas 37. Vi tänker oss att Alice ska skriva in en följd av tal, $a_1, a_2, \ldots a_n$, där varje a_i är ett tal mellan 0 och 36 i en dator (kanske på ett internetformulär). I praktiken är det kanske så att Alice skriver in en följd av siffror och bokstäver, och att datorn översätter till tal i bas 37 när den genomför kontrollen.

En följd $(a_1, a_2, \ldots, a_n)$ är ett kodord om

$$1 \cdot a_1 + 2 \cdot a_2 + \ldots n \cdot a_n = 0 \pmod{37}.$$

Exempel 4. Är $(0, 0, 7, 4)$ ett tillåtet kodord? Ja, för att

$$1 \cdot 0 + 2 \cdot 03 \cdot 7 + 4 \cdot 4 = 21 + 16 = 37 \equiv 0 \pmod{37}.$$

△

Om Alice skriver fel i en siffra eller byter plats på två intilliggande siffror kommer den koden upptäcka det. Det visar sig att det är viktigt för den egenskapen att 37 är ett primtal.