MAT1030 – Diskret matematikk

Forelesning 3: Mer om representasjon av tall

Dag Normann

Matematisk Institutt, Universitetet i Oslo

21. januar 2008

Oppsummering av Uke 3

Mandag 14.01 og delvis onsdag 16.01 diskuterte vi hva som menes med en algoritme, og vi så på pseudokoder og de tilhørende kontrollstrukturene.

Forventede ferdigheter er:

- Kunne uttrykke en algoritme i pseudokode, og kunne bruke de forskjellige kontrollstrukturene på riktig måte.
- Kunne følge en algoritme gitt ved en pseudokode og input- verdier på variablene skritt for skritt, og kunne holde orden på hvordan verdiene på variablene endrer seg under "utregningen".
- Kunne forklare hvorfor en pseudokode løser den oppgaven den er satt til å utføre.

Oppsummering av Uke 3

Onsdag snakket vi generelt om tall og tallsystemer, og mye om binær representasjon.

Vi la vekt på å beskrive algoritmer for overgang mellom tallene og deres binære representasjoner, både for heltall og desimaluttrykk.

Det gjenstår å utforme pseudokoden for algoritmen som finner binær desimal nr. k for et tall mellom 0 og 1.

Læringsmålene for dette kapitlet blir utformet senere, men de vil omfatte overganger til/fra binær representasjon både for heltall og for desimaltall.

Binære tall

Vi fortsetter med algoritmer i tilknytning til overgangen til/fra binære tall. Den neste pseudokoden vil definere en algoritme som gir oss siffer nr. k (fra høyre) til binærerepresentasjonen av et tall x slik at $0 \leq x < 1$. Hvorvidt **If-then-else**-testen er problematisk eller ikke, vil avhenge av hvordan vi representerer reelle tall som data.
Representasjon av tall som data kommer vi til i Kapittel 3.

1 Input x [$0 \leq x < 1$. x er det tallet vi vil finne binærformen til]

2 Input k [sifferet vi vil bestemme]

3 **For** $i = 1$ **to** k **do**

  3.1 **If** $x \geq \frac{1}{2}$ **then**

    3.1.1 $x \leftarrow 2(x - \frac{1}{2})$

    3.1.2 $z \leftarrow 1$

    else

    3.1.3 $x \leftarrow 2x$

    3.1.4 $z \leftarrow 0$

4 Output z

Binære tall

Vi illustrerer hvordan denne pseudokoden virker ved å la $x = \frac{1}{3}$ og la k være veldig stor.
For de første verdiene av i skriver vi ned hva den tilhørende verdien på x vil bli, og hvilken verdi z vil ha:

1. $x = \frac{1}{3} < \frac{1}{2}$ (før) så $\frac{2}{3} = x \leftarrow 2 \cdot \frac{1}{3}$ (etter). $z = 0$.
2. $x = \frac{2}{3} \geq \frac{1}{2}$ (før) så $\frac{1}{3} = x \leftarrow 2(\frac{2}{3} - \frac{1}{2})$ (etter), $z = 1$.
3. $x = \frac{1}{3} < \frac{1}{2}$ (før) så $\frac{2}{3} = x \leftarrow 2 \cdot \frac{1}{3}$ (etter). $z = 0$.
4. $x = \frac{2}{3} \geq \frac{1}{2}$ (før) så $\frac{1}{3} = x \leftarrow 2(\frac{2}{3} - \frac{1}{2})$ (etter), $z = 1$.

   ... ...

k $z = 0$ om k er et oddetall og $z = 1$ om k er et partall.

Aritmetikk

Vi utfører addisjon, subtraksjon, multiplikasjon og divisjon av tall på binær form omtrent som for tall i 10-tallsystemet, bortsett fra at alt i prinsippet blir mye enklere, den lille addisjonstabellen og den lille multiplikasjonstabellen blir så mye mindre.

Som eksempler regner vi følgende stykker på tavla (oppgaver for den som ikke er på forelesningen).

- $17 + 14$
- $17 - 14$
- $5 \cdot 11$
- $11 : 5$ med fire siffer bak komma.

Det er selvfølgelig mulig å finne pseudokoder som uttrykker de algoritmene vi vil bruke, men som i skolematematikken er det her best å demonstrere algoritmene ved eksempler.

OKTAL OG HEKSADESIMAL FORM

Hvis man bruker 8-tallsystemet arbeider man med tall på oktal form.
Eksempelvis vil vi ha

- $443_8 = 4 \cdot 8^2 + 4 \cdot 8 + 3 = 256 + 32 + 3 = 291_{10}$
- $3,21_8 = 3 + 2 \cdot \frac{1}{8} + \frac{1}{64}$

Hvis man bruker 16-tallsystemet arbeider man med tall på heksadesimal form.
Her må man supplere symbolene $1, \ldots 9$ med sifre A, B, C, D, E og F.

Eksempelvis vil

$$2C3_{16} = 2 \cdot 16^2 + 11 \cdot 16 + 3 = 512 + 176 + 3 = 691_{10}.$$

Oktal og heksadesimal form

Fordelen med oktal og heksadesimal form er at regning med tall i disse tallsystemene representerer en rasjonalisering av regning med binære tall.

Ved å gruppere tre og tre siffer kan en binær form omgjøres direkte til oktal form:

$$101\ 100\ 001\ 010_2 = 5412_8$$

og ved å gruppere fire og fire sifre kan en binær form omgjøres til heksadesimal form:

$$1011\ 0000\ 1010_2 = B0A_{16}.$$

Oktal og heksadesimal form

Heksadesimal form brukes i 16×16 sudokuer, og til å blande farver i HTML (256 styrkegrader av hver av de tre grunnfarvene), men oktale tall og heksadesimale tall ble brukt mer da man programmerte "nærmere" maskinspråket.

Vi skal ikke drille regning med tall på oktal eller heksadesimal form her.

Oktal og heksadesimal form

Oppgave (Tverrsumstest)

Gå tilbake til beviset for at tverrsumstesten for delelighet med 3 og 9 holder i 10-tallsystemet, og finn ut for hvilke tall vi har en tverrsumstest for tall på oktal og heksadesimal form.

OVER TIL KAPITTEL 3

REPRESENTASJON AV TALL I DATAMASKINER

For å forstå hvordan man lagrer tall i en datamaskin, må man vite litt om hvordan informasjon lagres generelt.

- Grunnenheten er en bit
- En *bit* vil være i en av to tilstander:
  - 0 eller 1.
  - Positiv eller negativ.
  - Sann eller usann.
  - PÅ eller AV.
  - $\cdots$ eller $\cdots$
- Vi vil systematisk bruke 0 og 1 som de former for informasjon en bit kan inneholde.
- Hvordan dette skjer fysisk i hver enkelt datamaskin vil vi ikke bry oss om.

Representasjon av tall i datamaskiner

- Passe store blokker av bits kalled bytes.
- Det normale er at en byte består av 8 bits, og at vi kan skrive informasjonen i en byte som eksempelvis

$$10011001$$

- Vi kan komme til å tillate oss å arbeide med fire- bits bytes, men det er for at det da blir lettere å forklare hva som skjer.

I en byte med 8 bits kan vi lagre $2^8 = 256$ forskjellige informasjoner. Dette svarer til alle tall med to sifre i det heksadesimale systemet.

Representasjon av tall i datamaskiner

Hvorfor skal vi lære om hvordan tall representeres i en datamaskin?

- Vi skal se at hele tall og reelle tall er forskjellige typer tall, og at "samme" tall må representeres forskjellig når det oppfattes som heltall og når det oppfattes som reelt tall.
- Ved å kjenne til hvordan tall representeres, vil vi kjenne til begrensningene og mulige feilkilder. Hvis man skal foreta en numerisk beregning hvor antall avrundinger er i millionklassen, er det viktig å vite hvor stor feil som kan oppstå fordi representasjonen i maskinen ikke er nøyaktig. Det finnes uendelig mange tall, men bare endelig mange av dem kan representeres i en konkret maskin.
- De som utdannes fra UiO som landets fremtidige dataeksperter, bør ha generelle kunnskaper om grunnlaget for det de er eksperter på.

Representasjon av hele tall

Når vi skal representere hele tall i en datamaskin er det tre spørsmål som må besvares:

1. Hvor mange tall ønsker vi å representere?
2. Hvilke tall ønsker vi å representere?
3. Hvordan vil vi representere dem?

Svaret på spørsmål 1 er et spørsmål om hvor mange bits/ bytes vi vil sette av for å representere et enkelt tall.
Bruker vi flere bits, kan vi representere flere enkelttall, men vi vil bruke lengere tid på å manipulere tallene, og vi vil ha mindre plass til andre formål.

Representasjon av hele tall

Som eksempel bruker boka 16 bits fordelt på to bytes.
Det gir at vi kan representere $2^{16} = 65536$ forskjellige tall.

Det normale er å bruke flere bits, eksempelvis 32 bits, i ordentlige maskiner.
For lettere å vise eksempler og forklare fenomener, skal vi begrense oss til "lekemaskiner" hvor vi bruker 8 bits.
Det gir oss bare muligheten til å representere 256 tall. Prinsippene vil være de samme.

Representasjon av hele tall

Som svar på spørsmål 2, Hvilke tall ønsker vi å representere? skal vi bygge på erfaringen at det er gunstig å representere et segment av tall-linjen med omtrent like mange tall på den positive og negative siden.

Siden vi trenger en bit til å bestemme om tallet er negativt eller ikke (det vil si fortegnet), kan vi representere $2^7 = 128$ tall med fortegn $+$ og 128 tall med fortegn $-$.

Det betyr at i vår lekemaskin skal vi kunne representere alle heltall a slik at

$$-128 \leq a \leq 127.$$

Representasjon av hele tall

Hvordan skal vi så representere disse tallene?

Noen valg må gjøres bare fordi vi må treffe et valg, mens andre valg peker seg umiddelbart eller etterhvert ut som de mest praktiske.

I det valget vi gjør legger vi mest vekt på følgende kriterium:

- Vi vil kunne overføre mest mulig av aritmetikken for tall på binær form til operasjoner på representasjonene.

Den første konsekvensen vi trekker av dette er at vi bør bruke første bit til å representere fortegnet, ettersom binær addisjon foregår fra høyre mot venstre.
Den andre konsekvensen er at vi bør la 0 i første bit representere $+$, og i det tilfellet la resten av de 7 bit-ene gi oss binærrepresentasjonen av tallet. Da vil alle tallene fra 0 til 127 bli representert ved sin binære form med en ekstra ledende 0.

Representasjon av hele tall

Det kunne vært naturlig å representere et negativt tall, eksempelvis -17 som 1 og så binærformen til 17 (med 7 sifre) etterpå.
Det er to grunner til at vi vil gjøre det anderledes:

- Vi ville få to representasjoner av 0 og ingen av -128.
- Vi ville ikke kunne brukt samme prosedyrer for addisjon og subtraksjon for positive og for negative tall.

Vi skal lage en prosedyre for å trekke fra 1, og som virker for representasjon av tall > 0, og vi skal se på hva slags representasjoner vi da får hvis vi følger prosedyren over i de negative tallene.

Representasjon av hele tall

Eksempel (Trekk fra 1)

1 Input $x_1 \cdots x_8$[Representasjonen av tallet]

2 $i \leftarrow 8$

3 **While** $i > 0$ **do**

  3.1 **If** $x_i = 1$ **then**

    3.1.1 $x_i \leftarrow 0$

    3.1.2 $i \leftarrow 0$

    else

    3.1.3 $x_i \leftarrow 1$

    3.1.4 $i \leftarrow i + 1$

4 Output $x_1 \cdots x_8$

Representasjon av hele tall

Eksempel (Noen inputrepresentasjoner)

- Representasjonen av 97 ($= 64 + 32 + 1$) vil være 01100001, og gir vi dette som input i pseudokoden får vi 01100000, som er representasjonen av 96 ($64 + 32$).
- Bruker vi 01100000 som input, får vi 01011111, som representerer

$$95 = 64 + 16 + 8 + 4 + 2 + 1.$$

- Bruker vi representanten for 0, 00000000, som input, får vi 11111111 som svar.
- Det kan derfor være naturlig å la 11111111 representere -1

Representasjon av hele tall

- Vi lar altså byten hvor 1 står i første bit, og binærrepresentasjonen av $127 = 128 - 1 = 128 + (-1)$ i resten av bit-ene, representere -1.
- Bruker vi pseudokoden vår på 11111111, får vi 11111110, og det bør være representasjonen av -2.
- Dette er det samme som 1 etterfulgt av binærrepresentasjonen av $126 = 128 - 2 = 128 + (-2)$
- Vi ser at hvis n er et tall ekte mellom 0 og 129, så peker 1 etterfulgt av binærrepresentasjonen til $128 + (-n)$ seg ut som en egnet representasjon av $-n$