Föreläsning 7 - Automater, textsökning, tillstånd

Automater

En portkodsautomat med nio knappar kan se ut så här:
A B C
D E F
G H I
Anta att den rätta knappföljden är **DEG**. Då har automaten fyra olika *tillstånd*:

1. Starttillstånd.
2. Knapptryckning **D** har just gjorts.
3. Knapptryckningarna **DE** har just gjorts.
4. Knapptryckningarna **DEG** har just gjorts. Låset öppnas.

När automaten är i ett visst tillstånd och en viss knapp trycks ner övergår den i ett nytt tillstånd, och det kan beskrivas med en *övergångsmatris*:

```
  A B C D E F G H
1 1 1 1 2 1 1 1 1
2 1 1 1 2 3 1 1 1
3 1 1 1 2 1 1 4 1
```

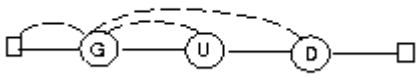
Exempel: Om automaten är i tillstånd 3 och knapp D trycks ner övergår den till tillstånd 2.

Man kan också rita en graf med fyra noder (som representerar tillstånden) och en massa bokstavsmärkta pilar (som visar vilka övergångar som finns).

Mer om automater får man veta i kursen 2D1373, Artificiella språk och syntaxanalys.

Textsökning

Samma automat kan användas för *textsökning*, till exempel för att söka efter **GUD** i bibeln. Bokstav efter bokstav läses och automaten övergår i olika tillstånd. När fjärde tillståndet uppnås har man funnit GUD. En berömd datalog, Donald Knuth, uppfann en enkel metod att konstruera och beskriva automaten. En Knuth-automat har bara en framåtpil och en bakåtpil från varje tillstånd. Så här blir den:



Ett nolltillstånd har skjutits in längst till vänster. Automaten startar emellertid i tillstånd 1, som har ett G i noden. Den tjuvtittar på första bokstaven i bibeln, och om det är ett G läser den G-et och går till höger. Annars följer den bakåtpilen utan att

glufsa bokstaven. I nolltillståndet glufsar den alltid en bokstav och går till höger. Koden blir i princip så här:

```
i=1
while (i<4):
   if  (i==0 or peek_next_char() == “adam”[i]) :
      i += 1
      next_char()
   else: i = next[i]
```

Här är gud[i] i-te bokstaven i det sökta ordet och next[i] det tillstånd man backar till från tillstånd i. Nextvektorn (bakåtpilarna) i vårt exempel blir

i	1	2	3
next[i]	**0**	**1**	**1**

Om vi i stället söker efter ADAM i bibeln blir Knuth-automaten så här:

Nextvektorn för ADAM blir alltså den här:

i	1	2	3
next[i]	**0**	**1**	**0**

För GUD gick bakåtpilen från tillstånd 3 till tillstånd 1, men här vore meningslöst att två gånger i rad kolla om bokstaven är A. Bakåtpilen från tillstånd 4 till tillstånd 2 kräver också en förklaring. Om vi har sett ADA och nästa bokstav inte är ett M kan vi i alla fall hoppas att det A vi just sett ska vara början på ADAM. Därför backar vi till tillstånd 2 och undersöker om det möjligen kommer ett D. Reglerna för hur nextvektorn bildas kan sammanfattas så här:

- next[1]=0.
- Annars är next[i]=1 om ordet inte upprepar sig.
- ...men om de j senaste bokstäverna vi sett bildar början på sökordet sätts next[i]=j+1.
- ...men om bokstav j+1 är samma som bokstav i sätts i stället next[i]=next[j+1].

Reglerna kan programmeras i några få satser och ger då den algoritm för textsökning som uppkallats efter Knuth, Morris och Pratt: KMP-automat. Om den sträng vi söker efter är m tecken lång och texten vi söker i är n tecken lång kräver KMP-sökning aldrig mer än n+m teckenjämförelser och är alltså O(n+m). Metoden går igenom texten tecken för tecken - man kan alltså läsa ett tecken i taget t.ex. från en fil vilket är praktiskt om texten är stor.

Tentafråga 19/10 2002

Rita en KMP-automat och ange nextvektor som söker efter ordet **VÅLNADSVÅLD**

Boyer-Moore

Då hela texten finns i en vektor kan man istället använda Boyer-Moores metod. Den börjar med att försöka matcha sista tecknet i söksträngen, som är *m* tecken lång. Om motsvarande tecken i texten inte alls förekommer i söksträngen hoppar den fram *m* steg, annars flyttar den fram så att tecknet i texten passar ihop med sista förekomsten i söksträngen.
Exempel: Vi söker efter TILDA i texten `MEN MILDA MATILDA`.

```
MEN MILDA MATILDA
TILDA
     TILDA
          TILDA
            TILDA
MEN MILDA MATILDA
```

Boyer-Moore är O(n+m) i värsta fallet, men ca n/m steg om texten vi söker i består av många fler tecken än dom som ingår i söksträngen, så att vi oftast kan hoppa fram m steg. När du skriver Ctrl-S för att söka efter en sträng i Emacs är det Boyer-Moore som används. Boyer-Moore använder en *skip*-vektor för att veta hur den ska hoppa framåt. Man kan läsa mer om algoritmer för textbehandling i kursen 2D1378, Text- och bildbehandling.

Sökning på webben

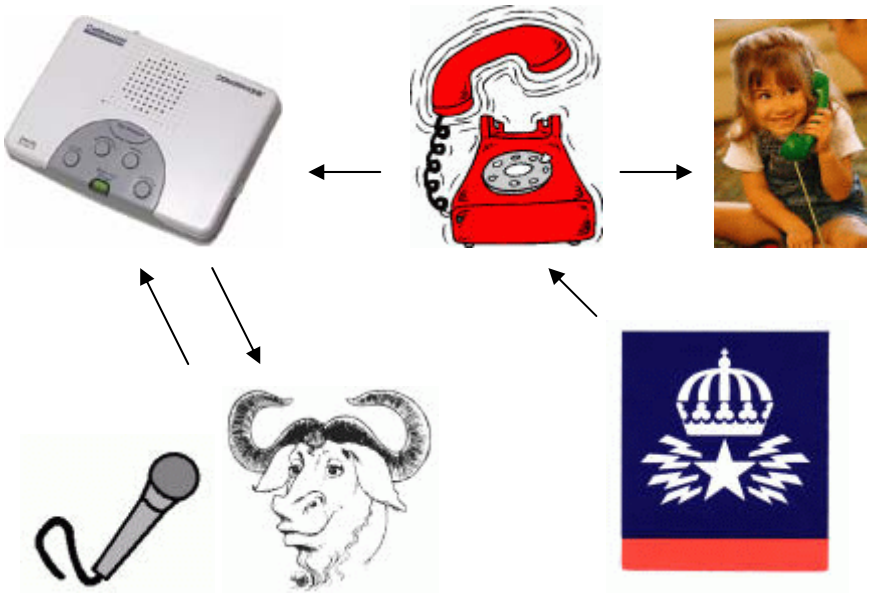
När man använder en *sökmotor*, t ex Google, för att hitta webbsidor som innehåller ett visst ord skulle alla ovanstående metoder bli för tidsödande. Där slår man istället upp ordet i ett index som skapats i förväg. Hur det fungerar kan man läsa mer om i kursen 2D1418, Språkteknologi.

Tillståndsprogrammering

Det finns flera problem som kan modelleras med olika tillstånd. Objektorienterad programmering lämpar sig väl för att implementera tillstånd. Antag att vi har en telefonsvarare med olika knappar.

I normalläge väntar telefonsvararen på ett samtal. Om det ringer ett visst antal signaler så svarar telefonsvararen och börjar eventuellt spela in ett meddelande. Trycker man på en inspelningsknapp så kan man spela in ett meddelande, då är telefonsvararen i ett svararläge.
Den här funktionaliteten kan vi också beskriva med en automat med olika *tillstånd, händelser (events)* och *aktiviteter (actions).* Nedan följer några förslag:

Tillstånd	Händelser	Actions
Vänta på signal Uppspelningsläge Inspelningsläge	Ringsignal Inspelningsknapp	Spela upp meddelande Spela in telefonsamtal



Det finns fler händelser och tillstånd, vad händer om telefonsvararen går igång men någon lyfter på luren och svarar? Vad händer om samtalet kopplas ner innan svararen spelat upp hela sitt meddelande? Hur smart är det med telefonsvarare med endast en knapp, kan man enkelt gå mellan olika tillstånd då?

Hur går det till rent praktiskt i programmeringen? Det finns olika sätt att lösa det men i princip används en medlemsvariabel som är åtkomlig från olika metoder. I vanliga fall bör metoder få all data via parametrar men i det här fallet ska man använda sig av en medlemsvariabel.

Telefonsvararen med en knapp, bankomater, grafiska gränssnitt, SAXparsning av XML-filer är andra problem som kan illustreras genom att rita en automat.

Objektmodellering och tillståndsprogrammering kan man läsa mer om i kursen 2D1385 Programutvecklingsteknik.

.