Föreläsning i ADK

Textsökning

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.

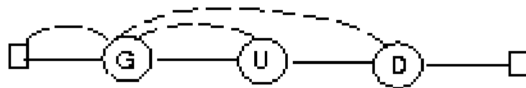
När automaten är i ett visst tillstånd och en viss knapp trycks ner övergår den i ett nytt tillstånd, och det kan beskrivas med en *övergångsmatris*:

	A	B	C	D	E	F	G	H		
1	1	1	1	2	1	1	1	1	1	Exempel: Om automaten är i tillstånd 3
2	1	1	1	2	3	1	1	1	1	och knapp D trycks ner övergår den till
3	1	1	1	2	1	1	4	1	1	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).

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. Datalogins fader, 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:



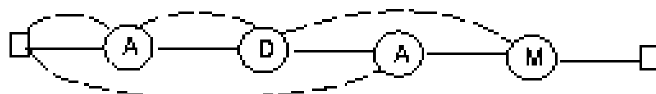
Ett nolltillstånd har skjutits in längst till vänster. Automaten startar emellertid i tillstånd 1, som har ett G i noden. Den tjuvtittar på första boktaven i bibeln, och om det är ett G läser den G-et och går till höger. Annars följer den bakåtpilen utan att glufsa bokstaven. I nolltillståndet glufsar den alltid en bokstav och går till höger. Koden blir i princip så här:

```
int i=1;
while (i<4) {
  if (i==0 || Mio.nextChar()==gud[i]) {
    i++;
    Mio.GetChar();
  }
  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 next[i]
1   0
2   1
3   1
```

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



Nextvektorn för ADAM blir alltså den här:

```
i next[i]
1   0
2   1
3   0
4   2
```

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]$\leftarrow j + 1$.
- ... men om bokstav $j + 1$ är samma som bokstav i sätts i stället next[i]$\leftarrow$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.

Horspools algoritm för sökning efter ordet P i strängen T

Horspool($P[1..m]$, $T[1..n]$)=
- **foreach** $c \in$ Alphabet **do** $Table[c] \leftarrow m$
- **for** $j \leftarrow 1$ **to** $m - 1$ **do** $Table[P[j]] \leftarrow m - j$
- $i \leftarrow m$
- **while** $i \leq n$ **do**
  - **if** $P[m] = T[i]$ **then**
    - **if** $P[1..m-1] = T[i-m+1..i-1]$ **then return** $i - m + 1$
  - $i \leftarrow i + Table[T[i]]$
- **return** ”ingen matchning”