Föreläsning 10-11

- Sökning och sortering
  - Sökning
    - Linjärsökning
    - Binärsökning
    - Binära sökträd
  - Sortering
    - Selection Sort
    - Quicksort
- Mer om typer i Modula-2
  - Procedurtyper
  - Mängdtyper

Sökning och sortering

Hantering av information

- Den vanligaste användningen av datorer.
- Stora mängder information kan behandlas mycket snabbt.
- Uppgifter:
  - Lagring av information
  - Åtkomst av lagrad information, sökning
  - Tillägg till och ändringar av lagrad information

Representation av information

- Val av lämpliga datastrukturer och algoritmer blir viktigare ju större datamängderna som ska hanteras är.
- Ofta har informationen en regelbunden struktur.
- Många liknande föremål i verkligheten ges liknande beskrivningar i datorn.
  - ett företags kunder
  - patienter på ett sjukhus
  - varor i ett lager
  - böcker i ett bibliotek

Sökning och sortering

Exempel

En typ för böcker

```
TYPE Name   = ARRAY [1..20] OF CHAR;
     Title  =ARRAY [1..50] OF CHAR;
     ISBN   = ARRAY [1..14] OF CHAR;
     LCcode = ARRAY [1..10] OF CHAR;

TYPE Book = RECORD
              author:Name;
              title:Title;
              publisher:Name;
              year:INTEGER;
              pages:CARDINAL;
              isbn:ISBN;
              lccode:LCcode;
            END;
```

- En bok tar upp 122 bytes.

Ett enkelt bokregister

```
CONST MaxAntal = 10000;

TYPE BokRegister =
        RECORD
          books:ARRAY [1..MaxAntal] OF Book;
          antal:[0..MaxAntal]
        END;
```

(Notera: detta är ett sätt att lagra en lista.)

- 10000 böcker tar upp ungefär 1Mb

Sökning och sortering

Ett enkelt sätt att lagra och söka böcker

En ny bok läggs till sist i listan

```
PROCEDURE newBook(b:Book; VAR reg:BokRegister);
BEGIN
  INC(reg.antal);
  reg.books[reg.antal]:=b
END newBook;
```

Att söka efter böcker

- Leta upp en bok med givet ISBN-nummer:

```
CONST NoBook= 0;
TYPE BookIndex = [NoBook..MaxAntal];

PROCEDURE findISBN(isbn:ISBN;
                   VAR reg:BokRegister):BookIndex;
VAR i:[1..MaxAntal];
BEGIN
  FOR i:=1 TO MaxAntal DO
    IF Compare(isbn,reg.books[i].isbn)=equal THEN
      RETURN i
    END
  END;
  RETURN NoBook
END findISBN;
```

- Böckerna ligger inte i någon särskild ordning, så man måste leta igenom alla från början till slut.
- Detta kallas *linjär sökning*.

Om resultat från sökningar

- Vid sökning efter ett ISBN-nummer hittar man högst en bok.
- Vid sökning på t ex författare kan man hitta många böcker.

  Använd en länkad lista av (index till) böcker för svaret.

Om linjär sökning

- Lätt att lägga till & ändra information.
- Söktiden blir proportionell mot antalet böcker.

  Antag att det tar 0.001s att testa om två ISBN-nummer är lika.

  Då kan det ta 100s att hitta en bok (av 100000)!

Ett exempel till

- Göteborgsdelen av telefonkatalogen innehåller ungefär 300000 nummer.
- Linjärsökning skulle kunna ta 300s!

  Snabbare att leta för hand.

  Hur fungerar nummerupplysningen?

- Om data är sorterade kan man söka mera metodiskt.

Spelet Hi-Lo

```
Välkommen till spelet HI-LO.
Jag tänker på ett tal mellan 1 och 100. Vilket? 50
50 är för stort, försök igen! 25
25 är för litet, försök igen! 37
37 är för stort, försök igen! 31
31 är för stort, försök igen! 28
28 är för litet, försök igen! 30
30 är korrekt!! Du behövde 6 gissningar.
```

Binärsökning

- Efter varje gissning halveras antalet möjliga svar.
- Med g gissningar kan man hitta rätt bland 2^g möjliga svar.
- Sambandet mellan antal möjliga svar n och antal gissningar g är alltså

$$n = 2^g$$

$$g = {}^2\log n$$

${}^2\log 100 = 6.6$

${}^2\log 100000 = 16.6$

${}^2\log 300000 = 18.2$

${}^2\log 8000000 = 22.9$

Binärsökning

```
PROCEDURE findISBN2(isbn:ISBN; VAR reg: BokRegister):
                                            BookIndex;

  PROCEDURE binSearch(first,last:CARDINAL):BookIndex;
  VAR middle:[1..MaxAntal];
  BEGIN
    IF first>last THEN
      RETURN NoBook
    ELSE
      middle:=(first+last) DIV 2;
      CASE Compare(isbn,reg.books[middle].isbn) OF
        less: RETURN binSearch(first,middle-1)
      | equal: RETURN middle
      | greater: RETURN binSearch(middle+1,last)
      END
    END
  END binSearch;

BEGIN
  RETURN binSearch(1,reg.antal)
END findISBN2;
```

Om binärsökning

- Informationen måste vara sorterad.
- Det krävs att alla element kan kommas åt lika lätt.

  En ARRAY fungerar bra.

  En länkad lista blir inte så bra.

Sortering

- En metod: se till att data är sorterade från början.

Insättning på rätt plats i en sorterad array

- ...om listan lagras i en array

```
TYPE BokRegister =
           RECORD
            books:ARRAY [1..MaxAntal] OF Book;
            antal:[0..MaxAntal]
           END;
```

- Metod:

  1. Leta upp rätt plats
  2. Flytta undan elementen efter rätt plats ett steg
  3. Sätt in det nya elementet

```
PROCEDURE newBook2(b:Book; VAR reg:BokRegister);
VAR i,pos:[1..MaxAntal];
BEGIN
  pos:=...; (* hitta rätt plats *)
  FOR i:=reg.antal TO pos BY -1 DO
    reg.books[i+1]:=reg.books[i]
  END;
  reg.books[pos]:=b;
  INC(reg.antal)
END newBook2;
```

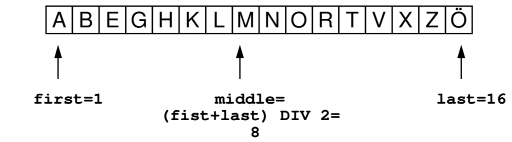
- Punkt 2 är den mest tidskrävande delen.

Sökning och sortering

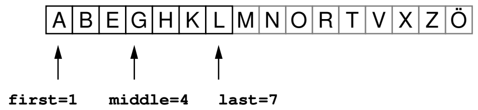
Binärsökning

Ett exempel

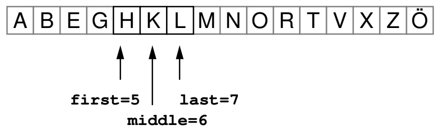
- Hitta "K"



- "K"<"M", forsätt med last=middle-1=7



- "K">"G", forsätt med first=middle+1=5



- Nästa steg om vi hade sökt "L": first=7, middle=7, klart.
- Nästa steg om vi hade sökt "J": last=5, ..., first=6. first>last.

Sökning och sortering

Om binärsökning

Fördelar

- Snabbt

  Antalet test för att hitta rätt bland n alternativ: ${}^2\log n$

Nackdelar

- Data måste vara sorterad.
- Data måste ligga i en ARRAY (så att alla element kan nås snabbt).

  Om man ska sätta in något nytt i en sorterad ARRAY måste allt som ska komma efter det nya elementet flyttas ett steg.

```
pos:=...; (* hitta rätt plats *)
FOR i:=reg.antal TO pos BY -1 DO
  reg.books[i+1]:=reg.books[i]
END;
```

  Långsamt! (I genomsnitt $n/2$ element måste flyttas.)

Observation

- Vid binärsökning undersöker man alltid elementet i mitten först, sedan elementet i mitten av första eller andra halvan.

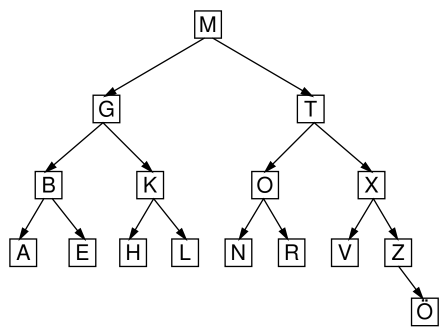
Sökning och sortering

Binära sökträd

- Ett träd som motsvarar

A	B	E	G	H	K	L	M	N	O	R	T	V	X	Z	Ö

och som går att söka i lika snabbt:



- Invariant för binära sökträd: för varje nod gäller:

  I vänster delträd finns bara element som kommer före i sorteringsordningen.

  I höger delträd finns bara element som kommer efter i sorteringsordningen.

Sökning och sortering

Om binära sökträd

Insättning i binära sökträd

- Nya element sätts in under ett löv. Att hitta rätt plats går fort.
- Länkad datastruktur: behöver inte flytta gamla data. (Jämför insättning på rätt plats i en länkad lista.)

Effektivitet

- Maximala antalet test som behövs för att hitta ett element = största djupet i trädet.
- För att sökning ska gå snabbt måste trädet vara balanserat. Största djupet i trädet blir då ${}^2\log n$.
- Trädets form beror på i vilken ordning data sätts in.

  Det första värdet som sätts in hamnar i roten. Nästa värde vänster eller höger barn till roten.

  Om data sätts in i sorterad ordning blir trädet skevt.

  Om data sätts in i slumpmässig ordning blir trädet ganska balanserat.

Sortering

Problemet

- Givet en array (med t ex bokstäver)

  V | L | K | G | R | Z | E | B | Ö | O | A | X | H | T | N | M

  ordna om elementen så att de kommer i en given ordning (t ex bokstavsordning).

  A | B | E | G | H | K | L | M | N | O | R | T | V | X | Z | Ö

```
PROCEDURE sortera(VAR a:ARRAY OF Element);
```

Två metoder

- En enkel med långsam: Selection Sort
- En komplicerad men snabb: Quicksort

Selection Sort

Idé

- Hitta det element som ska vara först, dvs det minsta elementet enligt ordningen.
- Byt plats på det element som ska vara först och det som råkade vara först.
- Fortsätt på samma sätt med resten av arrayen.

Exempel

V | L | K | G | R | Z | E | B | Ö | O | A | X | H | T | N | M

A | L | K | G | R | Z | E | B | Ö | O | **V** | X | H | T | N | M

A | **B** | K | G | R | Z | E | **L** | Ö | O | V | X | H | T | N | M

...

A | B | E | G | H | K | L | M | N | O | R | T | V | X | Ö | Z

A | B | E | G | H | K | L | M | N | O | R | T | V | X | **Z** | **Ö**

Om Selection Sort

Tidsåtgång

- Hur många jämförelser behövs för att sortera n element?

  Vi måste leta upp det minsta elementet n-1 gånger

  Första gången måste vi göra n-1 jämförelser.

  Sista gången räcker det med 1 jämförelse.

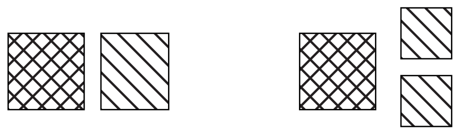
  I genomsnitt behöver vi göra n/2 jämförelser.

  Totalt behövs $(n-1)*n/2$ jämförelser, eller $(n^2-n)/2$.
- Hur många byten?

  Högst n-1 byten.

- För stora n blir tiden proportionell mot n^2.

Divide and Conquer (söndra och härska)



En problemlösningsmetod

- Dela upp problemet i två lika stora delar.
- Lös delproblemen var för sig.
- Sätt ihop dellösningarna till en lösning av hela problemet.

Kan det löna sig?

- Ja,

  om det går snabbare att lösa två problem av halva storleken än att lösa hela problemet

  och arbetet att dela upp problemet i delar och att sätta ihop dellösningarna inte är för stort.

- Sortering: hela problemet: n^2

  två häften så stora problem: $2(n/2)^2 = n^2/2$

Sökning och sortering

Quicksort

- En "divide-and-conquer"-metod

Updelning i två lika stora delproblem

- Välj ut ett element

V	L	K	G	R	Z	E	B	Ö	O	A	X	H	T	N	M

- Ordna om elementen så att alla som är mindre än det utvalda elementet kommer först, och alla som är större kommer sedan.

M	L	K	G	R	N	E	B	T	O	A	H	X	Ö	Z	V

Lösning av delproblemen

- Sortera delarna var för sig med Quicksort (rekursion).

A	B	E	G	H	K	L	M	N	O	R	T	V	X	Z	Ö

Ihopsättning av delresultaten

- Inget extra arbete!

Sökning och sortering

Om Quicksort

- Uppdelningen i delproblem blir inte perfekt.

Perfekt uppdelning

- Tiden blir proportionell mot $n \log n$

I värta fall

- Tiden blir proportionell mot n^2

I genomsnitt

- Tiden blir proportionell mot $n \log n$

Typer

Procedurtyper

(inklurerar funktionstyper)

Exempel

```
TYPE IntToInt  = PROCEDURE (INTEGER) : INTEGER;
TYPE IntToBool = PROCEDURE (INTEGER) : BOOLEAN;

TYPE BinOp = PROCEDURE (INTEGER, INTEGER) : INTEGER;

TYPE StringProc = PROCEDURE (ARRAY OF CHAR);
```

Procedurer som argument

```
PROCEDURE map(f:IntToInt; xs:List):List;

PROCEDURE filter(p:IntToBool; xs:List):List;
```

- Man saknar verkligen polymorfa typer...

```
TYPE RealFunc = PROCEDURE (REAL) : REAL;

PROCEDURE integrate(f:RealFunc; a,b:REAL):REAL;
PROCEDURE root(f,Df:RealFunc; x:REAL):REAL;
```


Typer

Procedurtyper

Procedurer som resultat

```
FROM MathLib IMPORT Sin, Cos, Tan;

TYPE TrigFunc = (sin, cos, tan);

PROCEDURE trigfunc(k:TrigFunc):RealFunc;
BEGIN
  CASE k OF
    sin: RETURN Sin
  | cos: RETURN Cos
  | tan: RETURN Tan
  END
END trigFunc;

...
```

Procedurvariabler

```
VAR f:RealFunc;
    x,y:REAL;

...

f:=trig(cos);
y:=f(x);
```

Typer

Procedurtyper

En begränsning

- I ML finns stegindelade funktioner:

```
fun add x y = x+y;

map (add 10) [1,2,3] = [11,12,13]
```

- Ett försök att göra samma sak i Modula-2

```
PROCEDURE add(x:INTEEGER):IntToInt;

  PROCEDURE addx(y:INTEGER):INTEGER;
  BEGIN
    RETURN x+y
  END addx;

BEGIN
  RETURN addx
END add;

map(add(10),cons(1,cons(2,cons(3,nil()))))
```

- Fungerar ej!
- Lokala procedurer kan ej användas som procedurvärden...
- Man kan inte tillverka nya funktioner under körningen.

```
PROCEDURE add10(y:INTEGER):INTEGER;
BEGIN
  RETURN 10+y
END add10;
```


Typer

Mängdtyper

Exempel

```
TYPE NumSet = SET OF [0..1000];
     LetterSet = SET OF ["A".."Z"];
     TrigFuncSet = SET OF TrigFunc;
```

- Elementtypen kan vara en godtycklig skalär typ (utom REAL).

Mängdkonstanter

```
NumSet{0,7,23}
NumSet{1,100..199,900..999}
LetterSet{"A","E","I","O","U","Y"}
```


Typer

Mängdtyper

Operationer på mängder

```
VAR a,b : SET OF ...
```

Operation	Modula-2
union	`a+b`
snitt	`a*b`
differens	`a-b`
symmetrisk differens	`a/b`

Jämförelser

```
= > < >= <= #
```

- Likhet: två mängder är lika om de innehåller samma element
- Ordning: en mängd är "mindre" än en annan om den är en delmängd

Mängdtillhörighet

```
x IN s
```

- Blir **TRUE** om **x** ingår i mängden **s**, annars **FALSE**.

Typer

Mängdtyper

Fler exempel

```
TYPE CharSet = SET OF CHAR;

CONST Letter = CharSet{"a".."z","A".."Z"};
      Digit = CharSet{"0".."9"}

      AlphaNum = Letter+Digit;
```

- Exempel: en procedur för att läsa in en identifierare

```
Ident = Letter { Letter | Digit }
```

...

-