Diskret matematikk – fredag 26/9 – 2014

Logiske matriser

En logisk matrise er en matrise der elementene er 0 (usann) eller 1 (sann). Dette kalles også en boolsk matrise. Læreboka kaller det en zero-one matrix.

Eksempler

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Husk de logiske operatorene $\neg$ (ikke), $\wedge$ (og), $\vee$ (eller).

1) $\neg 0 = 1, \quad \neg 1 = 0$
2) $1 \wedge 1 = 1, \quad 1 \wedge 0 = 0, \quad 0 \wedge 1 = 0, \quad 0 \wedge 0 = 0$
3) $1 \vee 1 = 1, \quad 1 \vee 0 = 1, \quad 0 \vee 1 = 1, \quad 0 \vee 0 = 0$

Logisk eller mellom to matriser (eng: join)

La A og B være to logiske $m \times n$-matriser (dvs. begge har samme dimensjon). Matrisen $A \vee B$ er den logiske matrisen vi får ved å ta parvis eller ($\vee$) mellom elementene i A og B. Elementet på plass i,j i $A \vee B$ er $a_{i,j} \vee b_{i,j}$.

Eksempel

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$A \vee B = \begin{bmatrix} 1 \vee 1 & 0 \vee 1 & 1 \vee 1 \\ 0 \vee 0 & 1 \vee 0 & 0 \vee 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Logisk og mellom to logiske matriser (eng: meet)

$A \wedge B$ defineres på en tilsvarende måte som $A \vee B$. Se forrige side.

Eksempel

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$A \wedge B = \begin{bmatrix} 1\wedge 1 & 0\wedge 1 & 1\wedge 1 \\ 0\wedge 0 & 1\wedge 0 & 0\wedge 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

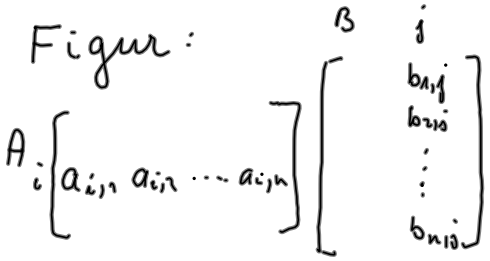
Logisk matrisemultiplikasjon

La A og B være to logiske matriser. Da er det logiske produktet $A \odot B$ definert hvis antall kolonner i A er lik antall rader i B. Det betyr at hvis A er en $m \times n$-matrise, så må B være en $n \times k$-matrise. Det logiske produktet $A \odot B$ blir da en $m \times k$-matrise

Elementet på plass i,j i $A \odot B$ er gitt ved:

$$(a_{i,1} \wedge b_{1,j}) \vee (a_{i,2} \wedge b_{2,j}) \vee \cdots \cdot (a_{i,n} \wedge b_{n,j})$$

Figur:



Elementet på plass i,j i $A \odot B$ er den "logiske multiplikasjonen" av rad i i A med kolonne j i B, dvs. gitt ved formelen over.

Eksempel

$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$, $A \odot B$ er ok siden A er en 2×3-matrise og B en 3×2-matrise, $A \odot B = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$

På skjemaform:

		B	
		1	0
		0	1
		1	0
A	0 1 1	x	y
	1 0 1	z	u

$x = (0 \wedge 1) \vee (1 \wedge 0) \vee (1 \wedge 1) = 1$
$y = (0 \wedge 0) \vee (1 \wedge 1) \vee \ldots = 1$
$z = (1 \wedge 1) \vee \ldots = 1$
$u = (1 \wedge 0) \vee (0 \wedge 1) \vee (1 \wedge 0) = 0$

$$A \odot B = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

Her er også $B \odot A$ definert siden B er en 3×2-matrise og A en 2×3-matrise. Da blir $B \odot A$ en 3×3-matrise.

		A		
		0	1	1
		1	0	1
B	1 0	x	y	z
	0 1			
	1 0			

$x = (1 \wedge 0) \vee (0 \wedge 1) = 0$
$y = (1 \wedge 1) \vee \ldots = 1$
$z = (1 \wedge 1) \vee \ldots = 1$
osv.

$$B \odot A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Kvadratiske logiske matriser

En kvadratisk logisk matrise kan (logisk) "multiplisres" med seg selv. Da kan vi skrive

$A^{[2]}$ istedenfor $A \odot A$, $A^{[3]}$ istedenfor $A \odot A \odot A$, osv.

Vi har $A^{[1]} = A$ og $A^{[0]} = I$ der I er identitetsmatrisen.

Kapittel 3 fra læreboken

NB!
Kun dette korte utdraget er pensum.

Algoritmer og algoritmeanalyse

En algoritme er en oppskrift på hvordan man løser et problem. "Oppskriften" kan formuleres med vanlig tekst, med såkalt pseudokode eller ved å bruke et programmeringsspråk. I Java brukes begrepene metode og algoritme delvis om hverandre.

Java-kode for logisk matriseprodukt

```
public class Program
{
  public static int[][] logiskMatriseProdukt(int[][] A, int[][] B)
  {
    if (A[0].length != B.length)
      throw new IllegalArgumentException("Ulovlige dimensjoner!");

    int m = A.length, n = B.length, k = B[0].length;

    int[][] C = new int[m][k];

    for (int i = 0; i < m; i++)
    {
      for (int j = 0; j < k; j++)
      {
        int resultat = 0;

        for (int p = 0; p < n; p++)
          resultat |= A[i][p] & B[p][j];

        C[i][j] = resultat;
      }
    }
    return C;
  }

  public static void main(String... args)
  {
    int[][] A = {{0,1,1},{1,0,1}};
    int[][] B = {{1,0},{0,1},{1,0}};

    int[][] C = logiskMatriseProdukt(B, A);

    for (int i = 0; i < C.length; i++)
    {
      for (int j = 0; j < C[0].length; j++)
      {
        System.out.printf("%4d",C[i][j]);
      }
      System.out.printf("\n");
    }
  }

} // class Program
```

Eksempel 1 Problem/oppgave: Finn den minste verdien i en heltallstabell.

En "oppskrift" i form av Javakode:

```
public static int minst(int[] a)   // en heltallstabell
{
  if (a.length == 0) throw
    new NoSuchElementException("Tabellen er tom!");

  int minstVerdi = a[0];   // foreløpig minst verdi
  for (int i = 1; i < a.length; i++)   // starter med i = 1
  {
    if (a[i] < minstVerdi) minstVerdi = a[i];  // oppdaterer
  }

  return minstVerdi;   // den minste verdien
}
```

Algoritmeanalyse

Algoritmeanalyse går ut på å analysere effektiviteten (mhp. tid og plass) til algoritmer.

Eksempel 2 La tabellen i metoden minst ha lengde n, dvs. $n = a.length$. Et mål på effektiviteten kan være antallet ganger sammenligningen a[i] < minstVerdi utføres. Vi ser at det blir $n-1$ ganger siden for-løkken starter med $i=1$. Vi sier derfor at denne metoden har orden n.

Eksempel 3

Sortering Vi får en enkel sorteringsmetode hvis vi gjentar ideen fra minst-metoden. Vi finner først den minste i tabellen. Så bytter vi om slik at den havner forrest i tabellen. Så finner vi den minste

av resten (dvs. den nest minste). Så bytter vi om slik at den havner nest forrest. Osv. En slik teknikk kalles utvalgssortering. Den kan kodes slik i Java:

```
public static void utvalgssortering(int[] a)  // en heltallstabell
{
  for (int i = 0; i < a.length; i++)  // yttre for-løkke
  {
    int minstVerdi = a[i];            // foreløpig minst verdi
    int m = i;                        // indeks til verdien
    for (int j = i + 1; j < a.length; j++)  // indre for-løkke
    {
      if (a[j] < minstVerdi)          // sammenligner
      {
        minstVerdi = a[j];            // oppdaterer minst verdi
        m = j;                        // oppdaterer indeksen
      }
    }
    // bytter om a[i] og a[m]
    int temp = a[i];
    a[i] = a[m];
    a[m] = temp;
  }
}
```

Algoritmeanalyse for utvalgssortering

Også her er det antallet ganger sammenligningen a[j] < minstVerdi utføres som er av interesse. La tabellen a ha lengde n. Første gang ($i=0$) blir det $n-1$ ganger siden den indre for-løkken da starter med $j=1$. Neste gang ($i=1$) starter den indre for-løkken med $j=2$. Da blir det $n-2$ sammenligninger. Osv. Tilsammen blir det:

$$(n-1) + (n-2) + \cdots + 3 + 2 + 1$$

Dette er en aritmetisk rekke med sum lik $(n-1+1)(n-1)/2$

$= \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$. Det viktige leddet her er n^2.

Derfor sier vi at utvalgssortering har orden n^2.

Flg. tabell viser de mest vanlige funksjonene i forbindelse med det å finne en algoritmes orden.

Utvalgssortering har orden n^2. De beste sorterings-metodene har lineæritmisk orden, dvs. orden $n \log_2 n$.
Tabellen viser hvor stor forskjell det er på en algoritme av orden n^2 i forhold til en av orden $n \log_2 n$ hvis f.eks. $n = 1000$ (tabellen har 1000 verdier).

OBS Algoritmeanalyse er en viktig del av faget Algoritmer og datastrukturer.

Funksjonstype	**n = 1**	**n = 10**	**n = 100**	**n = 1000**	**Beskrivelse**
$f_1(n) = 1$	1	1	1	1	*konstant*
$f_2(n) = \log_2 n$	0	3,3	6,6	9,97	*logaritmisk*
$f_3(n) = \sqrt{n}$	1	3,2	10	31,6	*kvadratrot*
$f_4(n) = n$	1	10	100	1000	*lineær*
$f_5(n) = n \log_2 n$	0	33,2	664,4	9.965,8	*lineæritmisk*
$f_6(n) = n^2$	1	100	10.000	1.000.000	*kvadratisk*
$f_7(n) = n^3$	1	1.000	1.000.000	10 siffer	*kubisk*
$f_8(n) = 2^n$	2	1.024	31 siffer	302 siffer	*eksponensiell*
$f_9(n) = n!$	1	3.628.800	158 siffer	2568 siffer	*faktoriell*