LØSNINGSFORSLAG - KOMMENTARER til SENSOR
N.B. RETTELSE 23.05 og 26.05 pkt. 1e)

:UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i : IN 115
Eksamensdag : Lørdag 20 mai, 2000
Tillatte hjelpemidler : Alle trykte og skrevne

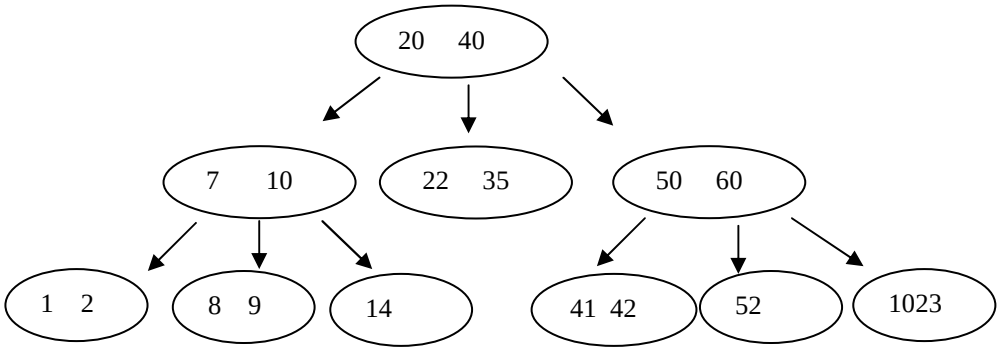
Arne Maus og Almira Karabeg

Totale testprogrammer for de to oppgaven, eksamensoppgaven og sensor-veiledning(denne) finnes på:

http://www.ifi.uio.no/~arnem/in115

Oppgave 1 (40 %)

Her var det en stygg 'trykkfeil' - nederst på s.1 stod det at :" . Det er bladnoder – dvs noder med bare tomme subtrær, som kan ha bare en verdi" . Dette ble opplyst tidlig på eksamen til alle at det måtte rettes til:". Det er bladnoder – dvs noder med bare tomme subtrær, som kan ha en eller to verdier." . Vi tror at ingen ble plaget av dette da eksempelet hadde både en og to verdier i bladnodene:



Oppgave 1a) Fellen her er at man glemmer å bytte om verdiene hvis den nye verdien er mindre enn den gamle:

```
void setV2(int v)
  { if ( v < v1)
    {int t = v1;
     v1 = v;
     v2 = t;
     } else v2 = v;
     v2Filled = true;
  }
```

Oppgave 1b) Her kan man svare både iterativt og rekursivt (like bra). Rekursivt er en løsning (numValues brukes i fjern - ikke påkrevet i 'insert'.:

```
public boolean insert(int v){
    if (numValues == 0) {rota = new TriNode(v); return true;}
    else return insert(root(), v);
  }
  public boolean insert(TriNode n, int v)
  {  if ( v == n.v1 || ( n.v2Filled && v == n.v2)) return false;
  if ( ! n.v2Filled){ n.setV2(v);numValues++;}
     else if (v < n.v1 )
      {if ( n.vsub == null )
        {n.vsub = new TriNode (v);numValues++;}
        else insert ( n.vsub, v);
      }
     else if ( v < n.v2)
      { if (n.msub == null )
        {n.msub = new TriNode (v); numValues++;}
        else insert (n.msub,v);
      }
     else
      { if ( n.hsub == null )
        {n.hsub = new TriNode(v);numValues++;}
        else insert (n.hsub, v);
      }

      return true;

  } // end insert
```

Oppgave 1c) (OldN er en variabel som ikke nyttes her, men i fjern -metoden, og løser problemet med å finne den noden som peker på den bladnoden som vi evt. skal fjerne.)

```
public TriNode find ( int v) {oldN = rota; return find (root(),v); }
  public TriNode find (TriNode n, int v)
  {if ( n == null) return null;
   else {
     if ( v == n.v1 || ( n.v2Filled && v == n.v2)) return n;
     else if (v < n.v1 ) {oldN =n; return find ( n.vsub, v);}
     else if ( v < n.v2) {oldN = n;return find(n.msub,v);}
     else {oldN = n; return  find (n.hsub, v);}
   }
  }// end find
```

Oppgave 1d)

```
public void skrivSortert(){ skrivSortert(root());}
 public void skrivSortert(TriNode n)
   { if ( n != null)
      {
        skrivSortert(n.vsub);
        System.out.println (n.v1);
        skrivSortert(n.msub);
        if (n.v2Filled) System.out.println (n.v2);
        skrivSortert(n.hsub);
      }
   }
```

Oppgave 1e) RETTELSE - 23.05 og 26.0:

Per def er høyda på rota = 0 Høyden på et velballansert trinærtre med n noder er da eksakt:

$h = \log_3 (2n+1) - 1$

, og som for helt fulle trinære trær som det er spørsmål om her vil gi et heltall (med: 1,4,13,40,... noder), men siden de aller fleste nodene er bladnoder, vil også:

$h = \log_3 (2n)$ - rundet av nedover til nærmeste heltall,

og.

$h = \log_3 (n)$ - rundet av nedover til nærmeste heltall,

gi riktig resultat

Oppgave 1f) Rot-verdiene plasseres i a[0] og a[1]; De tre subtrærne (v1-verdien) til en node med v1 i a[i], finner vi i a[i*3+2], a[i*3+4] og i a[i*3+6]. v2-verdien følger i elementet etter v1.

Oppgave 1g) *Kan puffes*. Dette er et vanskelig punkt - og lang kode. Poenget her er å først oppdage at en, to eller tre subtrær kan være tomme, og erstatnings-verdi evt. må søkes i alle mulige subtrær til noden (hvis de/det prefererte er tomt), eller i et spesialtilfelle, i noden selv. Det andre man må oppdage er at ersatningsverdien ikke nødvendigvis ligger i en blad-node. Da må den verdien som bringes opp i treet for å erstatte den verdien vi vil fjerne, selv erstattes (rekursivt) - til vi henter en erstatningsverdi i en bladnode (og denne noden evt. fjernes)

```
public boolean fjern(int v)
    // fjern verdi 'v' og erstatt denne med verdi lenger ned i treet
    // (rekursivt) hvis noden 'n' 'v' er i, ikke er bladnode
    // Hvis 'n' er bladnode, fjern 'n' hvis den blir tom
   {
    TriNode n = find (root(),v);
    if ( n == null) return false;  // Verdien finnes ikke
    else if ( bladnode(n))
    {
      if (v == n.v1 ) n.v1 = erstatt (oldN,n,false);
      else n.v2 = erstatt (oldN,n, false);
    }
    else if ( v == n.v1 )
    {
      if ( n.vsub != null ) n.v1 = erstatt (n,n.vsub, false);
      else if ( n.msub != null) n.v1 = erstatt (n,n.msub, true);
      else
        { // n.hsub != null, fordi ikke bladnode
            n.v1 = n.v2; n.v2 = erstatt (n,n.hsub, true);
        }
     }
     else
     { // v == n.v2
        if ( n.hsub != null ) n.v2 = erstatt (n, n.hsub, true);
        else if ( n.msub != null) n.v2 = erstatt (n,n.msub, false);
        else
        { // n.hsub != null, fordi ikke bladnode
            n.v2 = erstatt (n,n.vsub, true);
        }
     }
```

```
   numValues --;
   return true;
} // end fjern


private void fjernNode (TriNode prev, TriNode n)
// fjern bladnode n utpekt av prev
{
    if      (prev.vsub == n ) prev.vsub = null;
    else if (prev.msub == n ) prev.msub = null;
    else if (prev.hsub == n ) prev.hsub = null;

} // end fjernNode




private int erstatt ( TriNode prev, TriNode n, boolean minste )
  // finn en erstatning for (implisitt v) i noden n != null
  //  minste signaliserer om vi søker etter minste eller største
  {
   if ( bladnode(n))
   // bladnode, finn erstatning og marker evt noden for fjerning
   { if (minste)
      { int i = n.v1;
          n.v1 = n.v2;
          if (!n.v2Filled)
             fjernNode(prev,n);
          else n.v2Filled = false;
          return i ;
      }
      else if (n.v2Filled)
      { n.v2Filled = false;
        return n.v2;
      }
      else
      { fjernNode(prev,n);
        return n.v1;
      }
    }
    else if (minste)
    // erstatt 'v' med minste i dette subtreet
    { if ( n.vsub != null) return erstatt(n, n.vsub,true);
      else { // bruk v1 og erstatt denne (n.vsub == null)
            int i = n.v1;
            if (n.msub != null)
            { n.v1 = erstatt ( n, n.msub, true);
  af          return i;
            }
            else
            { // vet n.hsub != null
              n.v1 = n.v2;
              n.v2 = erstatt (n, n.hsub,true);
              return n.v1;
            }
          }
     }
     else
     { // erstatt med 'største' i dette subtreet
       if (n.hsub != null) return erstatt (n, n.hsub, false);
        else { // bruk v2 og erstatt denne (n.sub == null)
              int i = n.v2;
              if ( n.msub != null)
               { n.v2 = erstatt (n, n.msub,false);
                 return i;
               }
               else
               { // vet at n.vsub != null
                 n.v2 = erstatt (n, n.vsub, true);
```

```
                return i;
              }
        }
      }
  } // end 'erstatt'

private boolean bladnode (TriNode n)
  { return n.vsub == null && n.msub == null && n.hsub == null;}
```

Oppgave 2

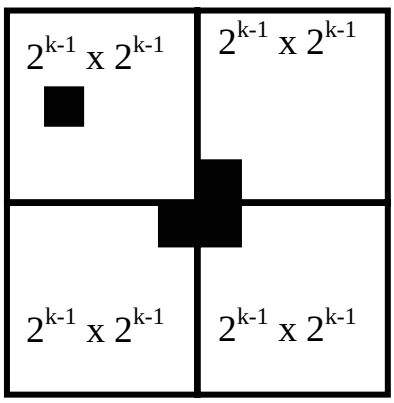
Oppgave 2a)

1. Det blir antall ruter minus 1 for den defekte ruten, delt på 3. Altså, svaret er $(2^{2k} - 1) / 3$.
2. Det er matematisk induksjon.

Oppgave 2b)

Generelt kan vi vel si at nyttes en annen, 'riktig' – men inneffektiv teknikk (som rekursjon med backtracing / perutasjoner) bør dette belønnes noe/litt selv om det blir inneffektivt

1. Teknikken som skulle brukes her er splitt og hersk. Man skal sette den første trimino i sentrum av det defekte sjakkbrettet, som vist på figur 3. Pass på at den plasseres slik at den ikke dekker den fjerdedelen av brettet som har defektiv rute. Da får man 4 subproblemer av størelsen $2^{k-1} \times 2^{k-1}$ som har akkurat en defekt rute hver. Se på tegningen under.



2. Når alle triominoer er plassert, begynner vi å fargelegge dem. Teknikken som gir en bra løsning her er grådig metoden.

   Vi kan her bruke to metoder for fargelegge - metode I) innser at man bare trenger 3 farger - mens II) - programmet - er en vanlig 'greedy' metode.

   I) Vi begynner med å gi fargen til triomino som dekker den øvre, venstre rute på brettet (hvis denne er den defektive ruten, fargelegger vi den ikke, men fortsetter med algoritmen). Den får farge 1. Deretter fargelegger vi triomino som dekker den neste ikke fargede (eller defektive) rute i første rad av brettet. Den er nabo triomino til den første fargete derfor får den farge nummer 2 (eller 1 hvis den første ruten var defekt). Vi fortsetter med å fargelegge den første ikke fargede rute (og resten av triomino som dekker den ruten) i øverste raden av brettet. Den er ikke lenger nabo til den første fargede triomino og kan få den samme fargen, dvs. farge nummer 1, osv. Man skulle observere her at en triomino generelt kan ha maksimalt 7 nabo triominoer og at det gir et konstant antall tester som må utføres for å finne hvilken farge som skulle brukes på den triominoen som fargelegges i øyeblikket.

II) - Oppgave 2 c)

```
public void tileAndColourBoard
       ( int topRow, int topCol, int defectRow, int defectCol, int size)
  { int half = size /2;
    int newRow = half +topRow;
    int newCol = half +topCol;


     if (size > 1)
       { // flere  partitions

         nextTile++;

         if ( (defectRow < newRow ) && (defectCol < newCol) )
          { // Ø.venstre kvadrant
              board[newRow ][newCol-1] = nextTile;
              board[newRow][newCol]    = nextTile;
              board [newRow-1][newCol] = nextTile;
              if(half > 1)
               { tileAndColourBoard(topRow,topCol,defectRow,defectCol,half);
                 tileAndColourBoard(newRow,newCol,newRow,newCol,half);
                 tileAndColourBoard(newRow,topCol,newRow,newCol-1,half);
                 tileAndColourBoard(topRow,newCol,newRow-1,newCol,half);
                }
          }
         else if ( (defectRow < newRow ) && (defectCol >= newCol ) )
          { // Ø. høyre kvadrant
              board[newRow -1][newCol-1] = nextTile;
              board[newRow][newCol-1]    = nextTile;
              board [newRow][newCol] = nextTile;
              if(half > 1)
              { tileAndColourBoard(topRow,newCol,defectRow,defectCol,half);
                tileAndColourBoard(topRow,topCol,newRow-1,newCol-1,half);
                tileAndColourBoard(newRow,topCol,newRow,newCol-1,half);
                tileAndColourBoard(newRow,newCol,newRow,newCol,half);
               }
```

```
        }
       else if ( (defectRow >= newRow ) && (defectCol < newCol ))
        { // N. venstre kvadrant
           board[newRow -1][newCol-1] = nextTile;
           board[newRow-1][newCol]    = nextTile;
           board [newRow][newCol] = nextTile;
           if(half > 1)
           { tileAndColourBoard(newRow,topCol,defectRow,defectCol,half);
             tileAndColourBoard(topRow,topCol,newRow-1,newCol-1,half);
             tileAndColourBoard(topRow,newCol,newRow-1,newCol,half);
             tileAndColourBoard(newRow,newCol,newRow,newCol,half);
           }
        }
       else if ( (defectRow >= newRow ) && (defectCol >= newCol ))
        { // N. høyre kvadrant
           board[newRow ][newCol-1] = nextTile;
           board[newRow-1][newCol-1] = nextTile;
           board [newRow-1][newCol] = nextTile;
           if(half > 1)
           { tileAndColourBoard(newRow,newCol,defectRow,defectCol,half);
             tileAndColourBoard(newRow,topCol,newRow,newCol-1,half);
             tileAndColourBoard(topRow,topCol,newRow-1,newCol-1,half);
             tileAndColourBoard(topRow,newCol,newRow-1,newCol,half);
           }
         }
     }

         // colour the tiles by the greedy method
        if ( size == lim )
        { // the first call
          // if (size== lim) return;
          for (int i = 0; i < lim ; i++)
           for (int j = 0 ; j < lim; j++)
            {
               int k = board[i][j];
               if( tileColour[k] == 0)
               { // this tile is not coloured
                 int colour =0;
                 do
                 { colour++;
                   tileColour [k] = colour;
                  } while (! okColour(colour,k,i,j));
                 if (colour > maxColour) maxColour = colour;
                }
            } // i,j
         }


}// end tileAndColourBoard

private boolean okColour(int colour, int tileNum, int i, int j)
{ int numFound = 1;

  for(int k= i-1; k < i+2; k++)
   for (int l= j-1; l < j+2; l++)
   { if ( (k>= 0 && k <lim && l>=0 && l<lim) && board[k][l] == tileNum)
     { // test neighbours of (k,l) that is tile 'tileNum' for same colour
        if (k-1 >= 0 && board[k-1][l] != tileNum )
           { if (tileColour[board[k-1][l]] == colour) return false;}
        if (l-1 >= 0 && board[k][l-1] != tileNum )
           { if (tileColour[board[k][l-1]] == colour) return false;}
        if (k+1 < lim && board[k+1][l] != tileNum )
           { if (tileColour[board[k+1][l]] == colour) return false;}
        if (l+1 < lim && board[k][l+1] != tileNum )
           { if (tileColour[board[k][l+1]] == colour) return false;}
      }
    }

    return true;
```

```
} // end okColour
```

Oppgave 2d) La T(k) være tiden som trengs til å flislegge det defekte 2^k x 2^k sjakkbrettet. Når k=0, size er lik 1 og konstant tid d trengs. Når k>0, fire rekursive kall er sett i gang. De tar 4 T(k-1) tid. I tillegg så er det if tester og flislegging av tre ikke defekte ruter. De tar konstant tid, la konstanten være lik c. Da får vi følgende rekursiv relasjon:

T(k) = d hvis k = 0, ellers

T(k) = 4 T(k-1) + c hvis k > 0.

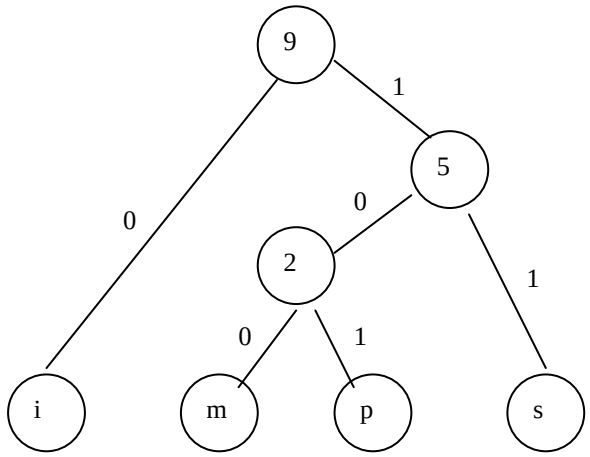
Den kan løses ved iterativ substituering: T(k) = 4 T(k-1) + c = 4 (4T(k-2) + c) +c = 4(4(4T(k-3) + c) + c) + c = ...= $O(4^k)$. Fargeleggings delen av algoritmen går gjennom hele brettet en gang, så den også tar $O(4^k)$ tid. Derfor hele algoritmen tar $O(4^k)$ og, faktisk, siden vi trenger $\theta(1)$ tid til å sette eller farge en flis, så er det hele også $\theta(4^k)$.

Oppgave 3

Oppgave 3a) Frekvenstabellen for ***isismispi*** er:

i	m	p	s
4	1	1	3

og da skal treet se slik ut:



Kodene for symboler er : i → 0, m → 100, p → 101 og s → 11.

Oppgave 3b) Dekoding av strengen gir: **misissipi.** Det som gjør at algoritmen alltid virker er at ingen kode er prefix av en annen kode. Algoritmen virker som følger:

1. Siffrene leses ett av gangen, fra venstre mot høyre.
2. Hvis sifferet som leses er en kode for en bokstav tildeles denne bokstaven til sifferet. Hvis ikke leses neste siffer og de siffer som er lest hittil sammenlignes mot kode-tabellen. Dette gjentas inntil en unik kode er funnet.
3. Når en kode er identifisert gjentas punkt to fra det første siffer etter den identifiserte koden inntil hele strengen er identifisert.