2004-03-12

Nim

En nyfiken resa in i tändstickornas matematiska värld

This report is about the game of Nim.
Nim was Invented in the early twentieth century by C.L. Bouton at Harvard University. The game is usually played with matches, which you put in a number of piles. Then the two players take turns removing matches. The player who removes the last match/es wins.

First you'll be shown how to model a game of Nim from any starting position. Then you'll be shown how to model a gametree from a few simple starting positions and how to determine who will win from any position in that tree. Then we'll present an induction proof of, that every position with two equally sized piles is lost for the player in turn.

Section 4 contains a finite-state automaton for winning a game with the starting position (4,4) where the automaton is player two.

In section 5 we present a computer program - "The Nimbler" which can play a game of Nim with three piles, each containing between 5-15 pieces. The programs strategy can be chosen by the player and ranges from randomly chosing every move to optimally playing the game from every position.

The last section describes a program which calculates the number of possible combinations of moves in which a game can be played, where the game is made up of three piles and a total of 45 game pieces.

Peter Alsbro
Michael Karling
Malte Martinsson
Tony Valcic
Jonas Österberg

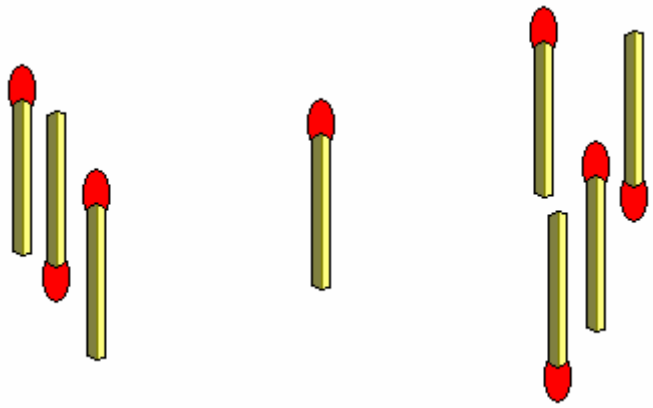
(1) Modellering och spelregler

Nim är ett spel som spelas med två spelare. För att spela Nim är allt man behöver något att använda som spelbrickor (oftast används tändstickor). Man delar upp tänsdstickorna i ett valfritt antal högar med ett valfritt antal stickor i varje. Därefter turas spelarna om att göra sina drag. Ett drag görs genom att man tar bort ett valfritt antal stickor (dock minst en) ur en och endast en valfri hög. Den som avlägsnar de sista stickorna/stickan har vunnit.

Vi ska nu titta närmare på hur man kan modellera ett parti Nim med valfritt antal högar:
n_1 = Antalet stickor i hög 1.
n_2 = Antalet stickor i hög 2.
o.s.v till:
n_k = Antalet stickor i den sista högen.
$0< x \leq n_?$ (x är antalet stickor som avlägsnas i ett drag och kan inte vara fler än antalet i den hög med flest antal stickor).

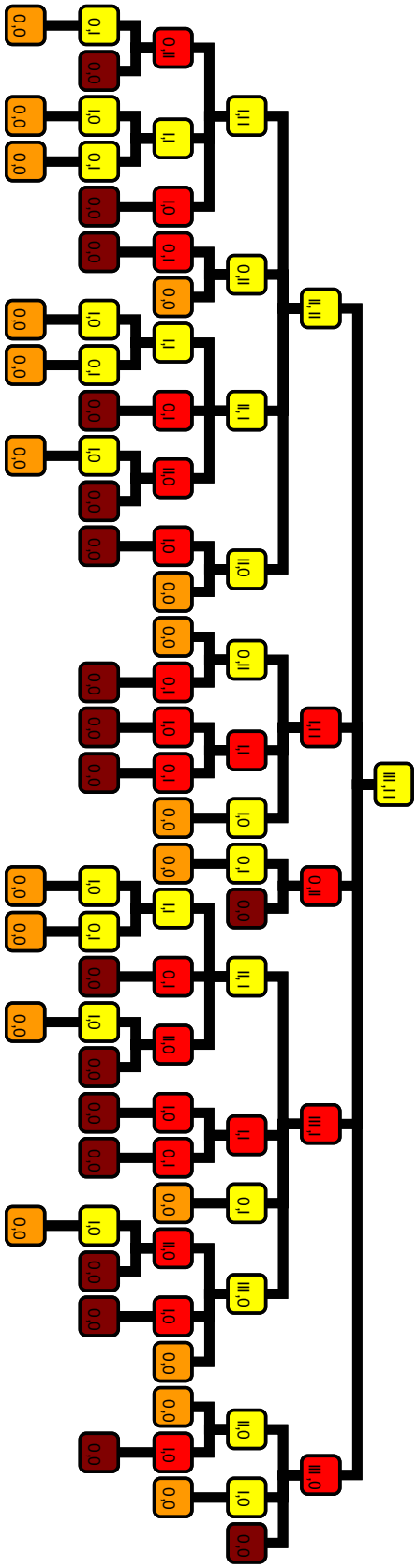
Varje given position i spelet kan alltså skrivas:
$(n_1, n_{2...}n_k)$ Där n är antalet stickor i en given hög och k är antalet högar.
Och varje drag kan skrivas på olika sätt beroende på vilken hög stickorna tas ifrån:
T.ex. $(n_1\text{-}x, n_{2...}n_k)$ Om stickorna tas från hög 1.

Vi kan nu modellera ett parti med två högar med vardera två stickor i:
(n_1, n_2) - Startposition (2, 2).
$(n_1\text{-}x, n_2)$ - Spelare 1 gör sitt drag och tar två stickor ur den första högen (2-2, 2).
(n_1, n_2) - Ny position (0, 2).
$(n_1, n_2\text{-}x)$ - Spelare 2 gör sitt drag och tar de sista stickorna (0, 2-2).
(n_1, n_2) - Spelare 2 vann (0,0).



Figuren ovan *- Ställning (3,1,4)*

(2) Spelträd från ställningen (3,2)



- Gula noder motsvarar en vunnen ställning för spelare ett.
- Röda noder motsvarar en vunnen ställning för spelare två.
- Oranga löv motsvarar vunnet spel för spelare ett.
- Bruna löv motsvarar vunnet spel för spelare två.

(3) Vem vinner i ställningen (n,n)?

Den familj av nim-ställningar som kan skrivas (n,n) är alla förlorade för den spelare vars otur det är att göra nästa drag, (spelare ETT). Vi bevisar här detta med hjälp av induktion.

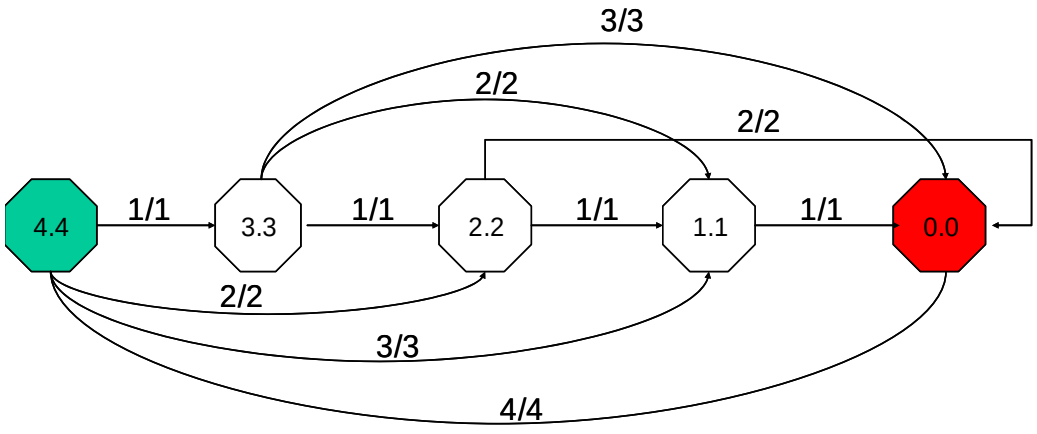
Basfall: Ställningen (0,0) är förlorad för spelare ETT.
Induktionssteg: Det antas att alla ställningar (m,m) där (0<=m<=n) är förlorad för ETT, då följer att (n+1,n+1) också är förlorad för ETT, pga följande.

ETT har att välja någon hög att minska men eftersom högarna är lika stora spelar det ingen roll vilken hon väljer. Säg att hon minskar den vänstra högen med d>0 tändstickor. Spelplanen blir då (n+1-d , n+1). Den riktigt smarta spelaren TVÅ ser då till att minska den andra högen lika mycket, vilket ger (n+1-d , n+1-d). Denna ställning ska ETT göra sitt drag ifrån, men eftersom n+1-d<=n så ser spelplanen ut som (m,m) där (0<=m<=n), vilket är en förlorad ställning för ETT, enligt förutsättning.
Alltså: Om alla ställningar från (0,0) till (n,n), (med de två högarna lika stora), är förlorade för ETT så är också (n+1,n+1) förlorad för ETT. Och eftersom (0,0) är förlorad för ETT så gäller det genom induktion att alla ställningar (n,n) (n>=0) är förlorade för ETT. V.S.V.

(4) Den listiga automaten

Vid startställning (4,4) i Nim kommer spelare Två alltid att vinna om han spelar som nedanstående automat.

Det är spelare Ett som ska göra sitt drag och spelare Tvås taktik är att alltid ta lika många stickor som spelare Ett, men från den hög som spelare Ett inte tog ifrån. Med denna taktik kommer endast 5 olika tillstånd finnas i automaten. De drag som kan göras är representerade med n/n där n kan vara ett tal mellan 1 och antalet stickor som finns kvar i en av högarna. Observera återigen att högarna alltid är lika stora med denna taktik. Högarna representeras som x.x i tillstånden där x är antalet stickor i en hög. Den gröna figuren är starttillståndet i automaten och den röda figuren är sluttillståndet.



Vi ser att automatens alla pilar representerar 2 drag. Först tar spelare Ett något följt av att spelare Två tar något. Vilken väg vi än tar från starttillståndet till sluttillståndet kommer den alltid att utgöras av ett jämnt antal drag vilket gör att spelare Två alltid vinner.

(5) Nim-programmets strategi

Bilaga nr 1 *är ett program som spelar nim med användaren. Programmets strategi är närmare knutet till en ekvivalent transformering av nim-spelet.*

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Spelplanen: Istället för en lista av heltal som representerar spelplanen kan man framställa den som matrisen till vänster. De fyra elementen i varje rad bildar den binära representationen av antalet tändstickor i högarna. Spelplanen till vänster har alltså tre högar (tre rader) och högstorlekarna är (1001b,1110b,0011b)=(9,14,3).

Reglerna: Enligt det ursprungliga nim-spelet så består ett drag av att välja en icke-tom hög och sedan välja hur många man ska ta bort från den högen. Med den finfina binär-matrisen till hands kan man istället helt ekvivalent säga så här.
Ett drag består av att först välja en 1:a i matrisen som sätts till 0, och sedan sätta vardera av elementen till höger på samma rad till vad man vill (0 eller 1).

De två dragen är ekvivalenta pga följande. Vi antar ett drag. Då man tar någon/några tändstickor från någon rad så minskas raden, vilket ger att den binära siffran längst till vänster av de bitar som ändras nödvändigtvis måste gå från 1 till 0, för annars skulle talet öka. Om man nu väljer just den siffran enligt de nya reglerna så får man ju sätta resten av siffrorna till höger till vad man vill, tex precis samma sak som subtraktionen gav enligt standardreglerna. Detta innebär att alla subtraktioner (högplockningar) kan göras med de nya reglerna. Vidare kan alla drag enligt de nya reglerna beskrivas som subtraktioner, detta eftersom den av siffrorna som ändras längst till vänster enligt nya reglerna måste gå från 1 till 0, vilket gör att talet måste minskas oavsett vad vi sätter siffrorna åt höger till, och alla minskningar kan beskrivas som subtraktioner. Därmed, eftersom alla drag enligt standard-reglerna också kan göras med de nya, och tvärtom, är reglerna ekvivalenta.

Strategin: ***För varje drag, gör så här:***
Om *antalet ettor* i varje kolonn är *jämnt* så är positionen förlorad, försök fuska.
Annars, leta reda på den kolonn längst till vänster där antalet ettor är udda.
Leta sen reda på någon av raderna i kolonnen där det står en etta, och sätt den till noll enligt reglerna, sätt sen alla de element till höger på raden på så sätt att alla motsvarande kolonner får ett *jämnt* antal ettor. Nu har alla kolonner ett jämnt antal ettor, vilket är nyckeln till vinst.

Strategins duglighet: Då man gör ett drag på en matris med endast ett jämnt antal ettor i varje kolonn så måste nödvändigtvis minst någon av kolonnerna få ett udda antal ettor efter draget. Detta eftersom draget endast sker i en enda rad varvid minst något av bit-värdena i raden måste ändras, och därmed invertera den motsvarande kolonnens "uddhet". Detta innebär att då man har udda matris, (en matris med någon udda kolonn), och reducerar matrisen till jämn, så är man säker på att få en udda matris tillbaks efter motspelaren gjort sitt drag. Och man kommer därför, om man följer strategin, alltid få göra sina fortsatta drag på en udda matris. Men den förlorade positionen är jämn, (en matris med bara nollor). Därför kan man aldrig hamna där, varför man måste vinna.

(6) Hur många sätt kan man spela Nim på?

En normal tändsticksask innehåller mellan 45 och 60 tändstickor. Vi kan bestämma oss för att 45 stycken är lämpligt att räkna med. I ett parti Nim ska en spelare ta minst 1 tändsticka eller max alla från en valfri hög. För att förenkla uträkningen av antalet spelsätt antar vi att de 45 stickorna ska placeras ut i exakt 3 högar där alla högar innehåller minst 1 sticka. Högarna har i de kommande beräkningarna ingen inbördes ordning, dvs. (1,1,43) och (1,43,1) är samma typ av spel. Jag tror att Nimfantaster är beredda att hålla med. Hur många spelsätt finns det då?

Det antal tändstickor som vi har är 45. Dessa 45 kan delas upp i 3 högar på flera olika sätt. Varje 3-högsformation kan sedan spelas på ett antal olika sätt. Hur många olika sätt det går att spela på med varje 3-högsformation beror på hur stickorna är fördelade i de 3 olika högarna. Det sammanlagda talet spelsätt är alltså summan av alla spelsätt från de olika 3-högsformationerna.

Hur räknar man ut hur många olika spelsätt som en 3-högsformation kan spelas på? Här angrips problemet med en rekursiv funktion!

Basfall: När summan av de 3 högarna är mindre än eller lika med 1 så går det bara att spela på 1 sätt.

Summma_spel_sätt(Hög1, Hög2, Hög3) = 1 Om (Hög1 + Hög2 + Hög3 <= 1)

Rekursionsteg: Vi har tre olika högar med olika antal stickor. De olika högarna är Hög1, Hög2 och Hög3. Antalet spelsätt för denna formation är summan av alla de spelsätt som vi kan göra efter att vi gjort ett drag, dvs. efter vi har tagit en eller flera tändstickor från någon hög.

$$\text{Summma\_spel\_sätt}(\text{Hög1}, \text{Hög2}, \text{Hög3}) = \sum_{i=1}^{Hög1} \text{summa\_spel\_sätt}(\text{Hög1} - i, \text{Hög2}, \text{Hög3}) +$$

$$\sum_{j=1}^{Hög2} \text{summa\_spel\_sätt}(\text{Hög1}, \text{Hög2} - j, \text{Hög3}) + \sum_{k=1}^{Hög3} \text{summa\_spel\_sätt}(\text{Hög1}, \text{Hög2}, \text{Hög3} - k)$$

Genom att göra på det här viset bygger man upp ett träd där varje löv innehåller någon av följande ställningar (0, 0, 0), (0, 0, 1), (0 1 0) eller (1, 0, 0). Det är vägarna ner till löven som är de olika spelsätten. Ett exempel på ett sådant här träd finns i analysen för vem som vinner en (3, 2) uppställning tidigare i detta dokument.

För att klara av denna uträkning har ett C-program skrivits. Programmet tar fram alla olika kombinationer av 3-högar av n antal stickor, där n är ett heltal större än 2. Sedan summerar programmet ihop alla dessa kombinationers respektive antal spelsätt. 45 stickor ger oss på detta sätt 184262179053145428804501504 olika spelmöjligheter, ungefär 1.843 * 10^26 sätt. Tar vi sedan bort begränsningen med 3 högar så blir det fler spelsätt ändå. Tar man även hänsyn till högarnas inbördes ordning blir det ytterligare fler spelsätt.
Programmet och testkörning utgör bilaga nr 2.

Bilaga 1 – Program som spelar nim med användaren, enligt en fiffig strategi.

```
#include <stdio.h>

int las_in(char* fraga,int min, int max) {
    int svar;
    do {
        printf("%s (%i..%i) ",fraga,min,max);
        if (scanf("%i",&svar)==0) { svar=min-1;   char flush[100];   scanf("%100s",flush); }
    } while (svar<min || svar>max);
    return svar;
}

int main() {
    int dator_iq,i,vems_tur,uddhet,val_hog,nn,hogar[3];
    srand((unsigned int)time(NULL));

    printf("======================================== The Nimbler\n");
    dator_iq=las_in("Computer IQ",0,200);
    for(i=0;i<3;i++) hogar[i]=5+random() % 11;
    vems_tur=random()%2;

    do {
        if(vems_tur==0)printf("Computer Move.....");  else  printf("....Your Move.....");
        for(i=0;i<3;i++) printf("(%i)->%i   ",i,hogar[i]);  printf("\n");
        nn=0;for(i=0;i<3;i++)nn+=hogar[i];
        if(nn==0) {
            if (vems_tur==0)printf("\n\nGood Work!\n\n");  else  printf("\n\nNimbler Rulez!\n\n");
            return 0;
        }

        if (vems_tur==0) {              // Nu är det datorns drag
            uddhet=0;  for(i=0;i<3;i++)uddhet=uddhet^hogar[i];       // xor-a högstorlekarna
            if(uddhet==0 || dator_iq<random()%200) {     // förlorad position eller datorn spelar dum
                for(i=0;i<3;i++)
                    if (hogar[i]>0) { hogar[i]=random()%hogar[i];  break;  }
            }
            else {
                for(i=0;i<3;i++)        // sätt lämplig hög så att binär-matrisen blir jämn
                    if ((hogar[i]^uddhet)<hogar[i]) {  hogar[i]^=uddhet;  break;  }
            }
        }
        else {                      // Nu är det användarens drag
            do { val_hog=las_in("....Pile",0,2); }  while (hogar[val_hog]==0);
            hogar[val_hog]-=las_in("....Pick",1,hogar[val_hog]);
        }
        vems_tur^=1;
    } while (1);
}
```

Bilaga 2 – Program och testkörning för olika spelsätt av Nim

Detta är ett program för att räkna ut hur många sätt man kan spela ett parti Nim på.
Programmet fördelar det inmatade antalet tändstickor på 3 högar i olika kombinationer.

```
#include <stdio.h>
#include <math.h>

#define MAXTAL 50
double rakna_spel_satt(int hog_1, int hog_2, int hog_3);
double power(double number, double multiplier);

double alla_tal[MAXTAL][MAXTAL][MAXTAL];

int main(int argc, char *argv[]) {
  double summa_spel_satt = 0;
  double potens = 0;
  double del_summa = 0;
  int hog1, hog2, hog3, antal_stickor;
  int i, j, k;

  if (argc == 2) {

    for(i=0; i < MAXTAL; i++)
      for(j=0; j < MAXTAL; j++)
   for(k=0; k < MAXTAL; k++)
   alla_tal[i][j][k] = 0;

    hog1 = 1;
    hog2 = 1;
    antal_stickor = atoi(argv[1]);
    hog3 = antal_stickor - hog1 - hog2;

    while(hog1 <= hog2 && hog2 <= hog3){
      while (hog3 >= hog2) {
   del_summa = rakna_spel_satt(hog1, hog2, hog3);
   printf("Delsumma for (%d,%d,%d) = %.0f\n", hog1, hog2, hog3, del_summa);
   summa_spel_satt += del_summa;
   hog2++;
   hog3--;
      }
      hog1++;
      hog2 = hog1;
      hog3 = antal_stickor - hog1 - hog2;
    }

    i = 0;
    potens = summa_spel_satt;
    while(potens > (double) 10) {
      potens /= (double) 10;
      i++;
    }
    printf("Antal spel: %.0f = %0.10f * 10^%d\n\n", summa_spel_satt,
potens, i);
  }
  else
    printf("USAGE nim [antal_stickor]\n");
  return 1;
}
double power(double number, double multiplier){
```

```
  double sum = 1;

  while(multiplier > 0 ) {
    sum *= number;
    multiplier--;
  }
  return sum;
}

double rakna_spel_satt(int hog_1, int hog_2, int hog_3) {
  double summa = 0;
  int i;

  if(alla_tal[hog_1][hog_2][hog_3]) {
    summa = alla_tal[hog_1][hog_2][hog_3];
  }
  else {
    if ((hog_1 + hog_2 + hog_3) <= 1 ) {
      summa = 1;
    }
    else if ((hog_1 + hog_2) == 0) {
      summa = power(2, hog_3 - 1);
    }
    else if ((hog_2 + hog_3) == 0) {
      summa = power(2, hog_1 - 1);
    }
    else if ((hog_1 + hog_3) == 0) {
      summa = power(2, hog_2 - 1);
    }
    else {
      for (i = 1; i <= hog_1; i++) {
   summa += rakna_spel_satt(hog_1 - i, hog_2, hog_3);
      }
      for (i = 1; i <= hog_2; i++) {
   summa += rakna_spel_satt(hog_1, hog_2 - i, hog_3);
      }
      for (i = 1; i <= hog_3; i++) {
   summa += rakna_spel_satt(hog_1, hog_2, hog_3 - i);
      }
      alla_tal[hog_1][hog_2][hog_3] = summa;
    }
  }
  return summa;
}
```

Testkörning:

> nim 45

.

.

Antal spel: 1842621790531454288045015O4 = 1.8426217905 * 10^26

Bilaga 3 - Mötesrapport och ansvarsfördelning

Uppgifter och fördelning

Vi har valt projektuppgift 4 – *Spela med tändstickor!* Tabellen nedan visar hur uppgifterna varit fördelade under projektets gång.

Peter Alsbro	1-5
Michael Karling	1-5
Malte Martinsson	1-5
Tony Valcic	6-9
Jonas Österberg	6-9

Protokoll för möten

Följande tider för projektmöten var bokade i våra kalendrar.

2004-02-06
Uppdelningen av uppgifter inom projektet spikades. Kortare diskussion om lösningar uppstod.

2004-02-13
Mötet den här veckan utgick till förmån för I-rapporter. En kort lägeskontroll hade vi för att se om någon kört fast.

2004-02-20
Lösningar till flera av uppgifterna fanns på papper. Vi diskuterade ideer och hjälptes åt med de uppgifter vi fastnat på.

2004-02-27
Till det här mötet fanns lösningar på alla uppgifter. De sista små frågetecknen diskuterades och löstes. Under veckan som följde diskuterades alternativa lösningssätt med stort intresse.

2004-03-05
Lösningarna diskuterades igen. Alla problem och lösningar sammanfogas till ett dokument som vi gärna kallar rapport. Den här mötesrapporten sammanställs.

2004-03-12
Rapprorten lämnas in. Vägen leder till kåren…