Denne guide er oprindeligt
udgivet på Eksperten.dk

COMPUTERWORLD

Tilfældige tal

Denne artikel introducerer generering af tilfældige tal og viser lidt om hvad man kan og ikke mindst hvad man ikke bør bruge.

Den forudsætter at man har lidt programmerings erfaring og kender lidt til de sprog jeg viser eksemplerne i (C, Java, C#).

Skrevet den **14. Feb 2010** af **arne_v** I kategorien **Programmering / Generelt** | ★★★☆☆

Historie:
V1.0 - 18/04/2005 - original
V1.1 - 29/12/2008 - tilføj links
V1.2 - 12/02/2010 - smårettelser

Hvad er tilfældige tal

Ægte tilfældige tal er meget svære at generere. Dertil skal man bruge noget udefrakommende som er ægte tilfældigt. F.eks. radioaktiv spaltning af atomer.

Det er ekstremt besværligt (læs: dyrt) at bruge til programmer så i praksis genererer man altid det man kalder pseudo tilfældige tal.

Pseudo tilfældige tal er ikke spor tilfældige. De er 100% deterministiske. Men de har nogle egenskaber som ligner tilfældige tal så meget at de i langt de fleste tilfælde er lige så gode at bruge.

Alle nedenstående eksempler drejer sig om det man kalder uniformt fordelte tal d.v.s. at alle tal har samme sandsynelighed for at blive udtrukket.

Fornuftig brug af indbyggede tilfældige tal generatorer

Rng.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 10
#define K 100

int main()
{
    int i,rani;
```

```
    double ranx;
    /* initialize random generator */
    srand(time(NULL));
    for(i=0;i<N;i++)
    {
        /* generate random integer in the range from 0 (inclusive) to K-1
(inclusive) */
        rani = rand() % K;
        printf("%d\n",rani);
    }
    for(i=0;i<N;i++)
    {
        /* generate random double in the range 0.0 (inclusive) to 1.0
(exclusive) */
        ranx = rand() / (double)(RAND_MAX + 1);
        printf("%f\n",ranx);
    }
    return 0;
}
```

Rng.java

```
import java.util.Random;

public class Rng {
    public final static int N = 10;
    public final static int K = 100;
    public static void main(String[] args) {
        // initialize random generator
        Random rng = new Random();
        for(int i=0;i<N;i++) {
            // generate random integer in the range from 0 (inclusive) to K-1
(inclusive)
            int rani = rng.nextInt(K);
            System.out.println(rani);
        }
        for(int i=0;i<N;i++) {
            // generate random double in the range 0.0 (inclusive) to 1.0
(exclusive)
            double ranx = rng.nextDouble();
            System.out.println(ranx);
        }
    }
}
```

Rng.cs

```
using System;
```

```
public class Rng
{
    public const int N = 10;
    public const int K = 100;
    public static void Main(string[] args)
    {
        // initialize random generator
        Random rng = new Random();
        for(int i=0;i<N;i++)
        {
            // generate random integer in the range from 0 (inclusive) to K-1
(inclusive)
            int rani = rng.Next(K);
            Console.WriteLine(rani);
        }
        for(int i=0;i<N;i++)
        {
            // generate random double in the range 0.0 (inclusive) to 1.0
(exclusive)
            double ranx = rng.NextDouble();
            Console.WriteLine(ranx);
        }
    }
}
```

Uheldig brug af tilfældige tal generatorer

Er output fra ovenstående så "passende tilfældigt" ?

De er ikke perfekte, men de er rimeligt fornuftige.

GoodRng.Java

```
import java.util.Random;

public class GoodRng {
    public final static int N = 50000;
    public final static int K = 10;
    public static void main(String[] args) {
        Random rng = new Random();
        int[] one = new int[K];
        int[][] two = new int[K][K];
        int[] a = new int[N];
        for(int i=0; i<N; i++) {
            a[i] = rng.nextInt(K);
        }
        for(int i=0; i<N; i++) {
            one[a[i]]++;
        }
        int last = a[0];
        for(int i=1;i<N;i++) {
```

```
            two[last][a[i]]++;
            last = a[i];
        }
        for(int i=0; i<K; i++) {
            System.out.println(one[i]);
        }
        for(int i=0; i<K; i++) {
            for(int j=0; j<K; j++) {
                System.out.print(" " + two[i][j]);
            }
            System.out.println();
        }
    }
}
```

Output

```
4945
5018
5085
4980
4956
4992
4846
5030
5045
5103
436 505 513 521 455 497 503 483 521 510
545 508 525 493 514 480 431 464 509 549
488 504 544 486 504 503 484 507 536 529
500 488 524 509 494 490 492 476 515 492
506 517 523 464 463 521 503 506 462 491
506 512 448 498 517 514 493 486 510 508
478 468 486 493 471 469 459 534 516 472
481 515 505 471 507 519 503 542 479 508
505 492 507 531 511 492 476 502 481 548
500 509 510 514 520 507 501 530 516 496
```

[I vil se det her output mange gange i denne artikel. Først vises fordelingen af tal - og den skal selvfølgelig gerne være jævn. Derefter vises fordelingen af tal, når det forrige tal kendes - og den skal også gerne være jævn]

Men der skal ikke meget til at ødelægge de pæne egenskaber.

Reinitialising af algoritme for hvert tal. En rigtig klassiker.
Når en initialisering for alle tallene er tilfældig så må en initialisering for hvert tal da være endnu mere tilfældig.

BadRng1.java

```
import java.util.Random;

public class BadRng1 {
    public final static int N = 50000;
    public final static int K = 10;
    public static void main(String[] args) {
        int[] one = new int[K];
        int[][] two = new int[K][K];
        int[] a = new int[N];
        for(int i=0; i<N; i++) {
            Random rng = new Random(); // <---- initialize for every number
            a[i] = rng.nextInt(K);
        }
        for(int i=0; i<N; i++) {
            one[a[i]]++;
        }
        int last = a[0];
        for(int i=1;i<N;i++) {
            two[last][a[i]]++;
            last = a[i];
        }
        for(int i=0; i<K; i++) {
            System.out.println(one[i]);
        }
        for(int i=0; i<K; i++) {
            for(int j=0; j<K; j++) {
                System.out.print(" " + two[i][j]);
            }
            System.out.println();
        }
    }
}
```

Output

```
0
0
16282
170
33548
0
0
0
0
0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 16281 0 1 0 0 0 0 0
0 0 0 169 1 0 0 0 0 0
0 0 1 0 33546 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

Tallene er overhovedet ikke tilfældige. Problemet er at initialiseringen bruger tiden som basis for de tilfældige tal. Så forskellige random generatorer initialiseret indenfor samme millisekund vil normalt give samme tal-

Løsningen er som i GoodRng.java ovenfor kun at initialisere en gang.

Ofte er det en fordel at lave det som en class eller instans variabel:

```
public static final Random rng = new Random();
```

Uheldig skalering. Ofte har man allerede en metode som returnerer et tilfældigt tal og det kan man da bare skalere. Et tilfældigt tal skaleret må da også være tilfældigt.

BadRng2.cs

```
using System;

public class BadRng2
{
    public const int N = 50000;
    public const int K = 10;
    public static void Main(string[] args)
    {
        Random rng = new Random();
        int[] one = new int[K];
        int[,] two = new int[K,K];
        int[] a = new int[N];
        for(int i=0; i<N; i++) {
            a[i] = rng.Next(15) % K; // scale random numner 0..14 to 0..9
        }
        for(int i=0; i<N; i++)
        {
            one[a[i]]++;
        }
        int last = a[0];
        for(int i=1;i<N;i++)
        {
            two[last,a[i]]++;
            last = a[i];
        }
        for(int i=0; i<K; i++)
```

```
        {
            Console.WriteLine(one[i]);
        }
        for(int i=0; i<K; i++)
        {
            for(int j=0; j<K; j++)
            {
                Console.Write(" " + two[i,j]);
            }
            Console.WriteLine();
        }
    }
}
```

[det er sjældent så åbenlyst i koden som her, men problemet er helt det samme uanset at rng.Next(15) eventuelt er gemt væk i en metode i en anden klasse]

Output

```
6686
6678
6700
6610
6558
3274
3408
3415
3316
3355
913 878 911 864 888 470 449 426 457 430
833 918 877 911 920 450 421 493 429 426
908 873 890 918 882 414 488 443 427 457
882 909 868 851 853 420 454 452 453 467
894 884 925 861 888 430 431 439 412 394
453 414 456 423 415 202 226 204 227 254
462 470 444 461 419 210 237 253 220 232
493 461 422 434 449 228 252 226 217 233
408 425 480 447 399 211 236 237 235 238
440 446 427 440 445 239 213 242 239 224
```

Tallene er overhovedet ikke tilfældige. Problemet er at 10 ikke går op i 15 og at 10 er tæt på 15, hvilket gør at der bliver en meget skæv fordeling.

Hvis den første modulus værdi er tilpas stor i forhold til den sidste modulus værdi, så har fænomenet ingen praktisk betydning.

Bemærk at følgende variant ikke løser problemet:

```
using System;

public class BadRng2NotFix
{
    public const int N = 50000;
    public const int K = 10;
    public static void Main(string[] args)
    {
        Random rng = new Random();
        int[] one = new int[K];
        int[,] two = new int[K,K];
        int[] a = new int[N];
        for(int i=0; i<N; i++)
        {
            a[i] = (int)((rng.Next(15) / 15.0) * K); // scale random numner
0..14 to 0..9
        }
        for(int i=0; i<N; i++)
        {
            one[a[i]]++;
        }
        int last = a[0];
        for(int i=1;i<N;i++)
        {
            two[last,a[i]]++;
            last = a[i];
        }
        for(int i=0; i<K; i++)
        {
            Console.WriteLine(one[i]);
        }
        for(int i=0; i<K; i++)
        {
            for(int j=0; j<K; j++)
            {
                Console.Write(" " + two[i,j]);
            }
            Console.WriteLine();
        }
    }
}
```

Output

```
6685
3212
6714
3323
6680
3280
```

```
6736
3367
6653
3350
910 421 910 465 877 389 884 455 925 449
428 188 418 225 410 207 452 236 453 195
856 413 943 449 917 446 889 431 909 461
415 227 433 228 458 232 457 227 420 226
877 435 920 427 898 433 902 433 899 456
437 222 394 200 484 235 427 229 438 214
877 443 895 433 916 440 898 462 902 469
455 206 428 207 439 258 461 219 465 229
948 433 893 460 866 423 928 461 801 440
482 224 480 229 415 217 438 214 440 211
```

Løsningen er aldrig at arbejde videre på tilfældige tal som allerede er skaleret ned en gang.

Hvis man skal skalere den indbyggede RNG ned til noget, så bør trække et nyt tal hvis man får noget >= (MAX_RAND / K) * K.

Uheldig kombination af tilfældige tal. Hvis man har 2 gode random generatorer så må man da kunne kombinere dem til en endnu bedre.

BadRng3.cs

```
using System;

public class BadRng3
{
    public const int N = 50000;
    public const int K = 10;
    public static void Main(string[] args)
    {
        Random rng = new Random();
        int[] one = new int[K];
        int[,] two = new int[K,K];
        int[] a = new int[N];
        for(int i=0; i<N; i++)
        {
            a[i] = (rng.Next(K) + rng.Next(K)) / 2; // average of two random
numbers 0..9
        }
        for(int i=0; i<N; i++)
        {
            one[a[i]]++;
        }
        int last = a[0];
        for(int i=1;i<N;i++)
        {
            two[last,a[i]]++;
```

```
            last = a[i];
        }
        for(int i=0; i<K; i++)
        {
            Console.WriteLine(one[i]);
        }
        for(int i=0; i<K; i++)
        {
            for(int j=0; j<K; j++)
            {
                Console.Write(" " + two[i,j]);
            }
            Console.WriteLine();
        }
    }
}
```

Output

```
1532
3568
5473
7410
9506
8536
6476
4494
2499
506
35 116 178 225 310 253 208 124 67 16
101 256 386 552 645 612 468 343 170 35
173 407 614 814 1064 907 660 497 283 54
221 497 783 1042 1537 1266 975 642 365 81
292 664 1077 1404 1776 1684 1185 852 494 78
267 648 908 1298 1575 1463 1084 765 432 96
217 452 693 989 1202 1057 891 582 325 68
140 315 512 648 838 766 612 406 215 42
75 183 261 370 461 430 329 232 126 32
11 30 61 68 98 98 64 51 21 4
```

Tallene er overhovedet ikke tilfældige. Det bliver en meget skæv fordeling.

Det kræver en meget fin forståelse for matematik at lave noget der kan kombinere 2 random generatorer til en.

[artiklen http://www.eksperten.dk/guide/686 "Mere om tilfældige tal" har eksempler på sådanne kombinationer der giver en korrekt fordeling]

Løsningen i praksis er at undgå den slags kombinationer.

Brug af dårlige low bits. Man skal være meget forsigtig med at tage modulus med potenser af 2.

BadRng4.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 50000
#define K 4

static unsigned long int seed;

void mysrand(unsigned long int ss)
{
    seed = ss;
    return;
}

unsigned long int myrand()
{
    /*
    seed = (65539UL * seed) % 2147483648UL;
    */
    unsigned long int help1 = 2147483648UL % 65539UL;
    unsigned long int help2 = 2147483648UL / 65539UL;
    long int tmp = 65539UL * (seed % help2) - help1 * (seed / help2);
    if(tmp >= 0)
        seed = tmp;
    else
        seed = tmp + 2147483648UL;
    return seed;
}

int main()
{
    int i,j,last,one[K],two[K][K],a[N];
    mysrand(time(NULL));
    for(i=0;i<N;i++)
    {
        a[i] = myrand() % K;
    }
    for(i=0; i<K; i++) one[i] = 0;
    for(i=0; i<N; i++) one[a[i]]++;
    for(i=0; i<K; i++) for(j=0; j<K; j++) two[i][j] = 0;
    last = a[0];
    for(i=1;i<N;i++)
    {
        two[last][a[i]]++;
        last = a[i];
```

```
    }
    for(i=0; i<K; i++) printf("%d\n",one[i]);
    for(i=0; i<K; i++)
    {
        for(j=0; j<K; j++) printf(" %d",two[i][j]);
        printf("\n");
    }
    return 0;
}
```

[tænk ikke så meget over algoritmen - den er ikke god, men var ikke desto mindre meget anvendt for 30 år siden]

Output

```
0
25000
0
25000
0 0 0 0
0 0 0 24999
0 0 0 0
0 25000 0 0
```

Tallene er overhovedet ikke tilfældige, hvilket skyldes at den algoritme genererer rimeligt tilfældige high bits men meget dårlige tilfældige low bits, og vi kigger jo kun på de 2 laveste bits.

Så en oplagt løsning er jo at bruge high bits.

BadRng4Fix.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 50000
#define K 4

static unsigned long int seed;

void mysrand(unsigned long int ss)
{
    seed = ss;
    return;
}
```

```
unsigned long int myrand()
{
    /*
    seed = (65539UL * seed) % 2147483648UL;
    */
    unsigned long int help1 = 2147483648UL % 65539UL;
    unsigned long int help2 = 2147483648UL / 65539UL;
    long int tmp = 65539UL * (seed % help2) - help1 * (seed / help2);
    if(tmp >= 0)
        seed = tmp;
    else
        seed = tmp + 2147483648UL;
    return seed;
}

int main()
{
    int i,j,last,one[K],two[K][K],a[N];
    mysrand(time(NULL));
    for(i=0;i<N;i++)
    {
        a[i] = (myrand() / 2147483648.0) * K;
    }
    for(i=0; i<K; i++) one[i] = 0;
    for(i=0; i<N; i++) one[a[i]]++;
    for(i=0; i<K; i++) for(j=0; j<K; j++) two[i][j] = 0;
    last = a[0];
    for(i=1;i<N;i++)
    {
        two[last][a[i]]++;
        last = a[i];
    }
    for(i=0; i<K; i++) printf("%d\n",one[i]);
    for(i=0; i<K; i++)
    {
        for(j=0; j<K; j++) printf(" %d",two[i][j]);
        printf("\n");
    }
    return 0;
}
```

Output

```
12628
12373
12597
12402
3171 3198 3192 3067
3079 3067 3055 3172
3191 3119 3213 3074
3187 2989 3137 3088
```

Man kan selvfølgelig også bruge en anden algoritme.

BadRng4Alt.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 50000
#define K 4

int main()
{
    int i,j,last,one[K],two[K][K],a[N];
    srand(time(NULL));
    for(i=0;i<N;i++)
    {
        a[i] = rand() % K;
    }
    for(i=0; i<K; i++) one[i] = 0;
    for(i=0; i<N; i++) one[a[i]]++;
    for(i=0; i<K; i++) for(j=0; j<K; j++) two[i][j] = 0;
    last = a[0];
    for(i=1;i<N;i++)
    {
        two[last][a[i]]++;
        last = a[i];
    }
    for(i=0; i<K; i++) printf("%d\n",one[i]);
    for(i=0; i<K; i++)
    {
        for(j=0; j<K; j++) printf(" %d",two[i][j]);
        printf("\n");
    }
    return 0;
}
```

Output

```
12671
12293
12526
12510
3233 3098 3119 3221
3022 3061 3139 3070
3270 3067 3105 3084
3146 3066 3163 3135
```

Men den første workaround er faktisk bedre medmindre man er meget sikker
på egenskaberne af den alternative algoritme man bruger.

Videre

Se artiklerne:
* http://www.eksperten.dk/guide/686 "Mere om tilfældige tal" som
  giver lidt flere eksempler, forklarer lidt teori og viser nogle anerkendte
  algoritmer
* http://www.eksperten.dk/guide/951 "Endnu mere om tilfældige tal" som forklarer lidt mere teori og har nogle
  eksempler i PHP og ASP

Kommentar af simonvalter d. 25. Apr 2005 | 1

meget interessant

Kommentar af md_craig d. 05. Aug 2006 | 2

""ægte tilfældigt. F.eks. radioaktiv spaltning af atomer.""
- der er mig bekendt ikke noget endegyldigt bevis for at det er tilfældigt... og jeg tror desuden på Tilfældighed som en definition på noget som ikke eksistere (Filosofisk)

""Hvis man har 2 gode random generatorer
så må man da kunne kombinere dem til en endnu bedre.""
- Vil våge den påstand med kendskab til sandsynligheds beregning, at outputtet her er i aller fineste orden... Man skal jo kende til sandsynligheds begreper

Kommentar af mysitesolution d. 27. Jun 2005 | 3

God artikle...

Har selv "funderet" over ordet tilfældighed, og er kommet til den konklusion at der er intet der er tilfældigt :/ (nu kender jeg ikke lige det med atom spaltning, så ved ikke om det er VIRKELIG tilfældigt, men tror det ikke), og er samtidig kommet til den overbevisning at vi mangler en mellemting mellem held og uheld, dvs. det neutrale.. :/ hmm

Kommentar af ofirpeter d. 20. Apr 2005 | 4

Kommentar af over-load d. 22. Apr 2005 | 5

=) Arne_v kvalitet

Kommentar af wicez (nedlagt brugerprofil) d. 19. Apr 2005 | 6

God artikel. Værd at læse, selvom man ikke kender til nogen af de 3 sprog er det meget godt baggrundsviden.

Kommentar af visualdeveloper d. 17. Oct 2005 | 7

god artikel ;)

Kommentar af hyberpreprocessor d. 19. Apr 2005 | 8

nice, gad vide hvad lotto systemet så bruger :D:D

Kommentar af phoenix2 d. 22. Apr 2005 | 9

Hvad er et falsk tilfældigt tal?