VJ Davey

# CSci 426 Midterm Exam One

1. **Problem 1:** [50 Points] According to the ANSI C Standard, the type unsigned long is guaranteed to support integers in the range $0, 1, ..., 2^{32} - 1$.

   (a) **Modify algorithm 2.2.1 such that it never produces an integer outside of the range** $\{0, 1.., m\}$**.** My main adjustment to algorithm 2.2.1 was making it so that all variables such as a, x, q, r, and m were of type *long long*. Doing this and compiling in c99 mode made it so numbers up to $2^{64} - 1$ could be handled by and generated by the Lehmer generator. My base code was the following, and I made simple adjustments to it in order to answer the succeeding questions, much like how we did for a previous homework assignment.

```
\\x, a,  and m are predefined long longs
const long long q = m / a;
const long long r = m % a;
long long t;
t = a * (x%q) - r * (x/q);
if (t > 0)
  return t;
else
  return( t + m );
```

   (b) **How many full period multipliers are there for** $m = 2^{32} - 135$**?**
   There are 67,181 full period multipliers

   (c) **For** $m = 2^{32} - 135$**, how many full period, modulus compatible multipliers are there? What are the ten smallest?**
   There are 15,201 full period modulus compatible multipliers.
   The 10 smallest ones are: $\{58, 72272, 741022, 57229, 37660, 223906, 849815, 66123, 75428, 32754\}$.

   (d) **Find the smallest full period, modulus compatible multiplier** $a \geq 100,000$ **for modulus** $m = 2^{32} - 135$**. For this (a,m) pair, construct a table of maximal modulus-compatible jump multipliers.(Similar to example 3.2.6) for 1024, 512, 256, 128, 64, and 32 streams.**

$$a = 741022$$

| *No.Streams* | $m/s$ | $j$ | $a^j \mod m$ |
|---|---|---|---|
| 1024 | 4194303 | 4174774 | 114977 |
| 512 | 8388607 | 8271613 | 33678 |
| 256 | 16777215 | 16677120 | 33487898 |
| 128 | 33554430 | 33487898 | 15913 |
| 64 | 67108861 | 67053787 | 79660 |
| 32 | 134217723 | 134194269 | 145203 |

2. **[25 points] A rod of length 1 is broken into 3 pieces, at random. (a) Use Monte Carlo simulation to estimate that the resulting three pieces form a triangle. (b) Explain your interpretation of "at random" (c) What is the axiomatic probability?**

   (a) My code for this is on the last pages of my exam. For randomly chosen seeds, my MC simulation gave the following estimates at 9999999 repetitions for each seed :

   | $x_0$ | $Triangles$ |
   |---|---|
   | 234 | 2498173 |
   | 343 | 2499403 |
   | 42 | 2500663 |
   | 893 | 2501012 |
   | 777 | 2501699 |

   From my MC Simulation results, I've deduced that a triangle is formed 25.002 % of the time.

   (b) My interpretation of "at random" is one person (one source/stream of randomness) randomly choosing two break points on a rod of length one.So we have a uniform selection from 0 to 1. Choosing two break points creates three pieces, and from there we can use triangle inequalities to determine if we have a triangle or not.

   (c) The Axiomatic probability is 25%. A triangle is possible if no piece is greater than half the length of the original rod by Triangle Inequalities. Due to uniform selection, there is a 50% chance for both break points to end up on the same side of the middle point. Compounding onto this, there is another 50% chance that other break point ends up placed far enough into its piece of the rod that it creates a middle piece greater than half the length of the rod. Therefore we have $\frac{1}{2} * \frac{1}{2} = \frac{1}{4}$

3. **[25 points] The two-player game of tic tac toe is played on a 3x3 board where players "X" and "O" take turns writing their character on the board. A player wins if she has three of her characters in a line. The game ends when a player wins or there are no empty spaces on the board. Player "X" always goes first. Assume both players play at random.**

   (a) **Write a Monte Carlo Simulation to estimate the probability that player "X" wins, player "O" wins, and nobody wins.**
   My code for this is on the last pages of my exam. For randomly chosen seeds, my MC simulation gave the following estimates at 9999999 repetitions for each seed :

   | $x_0$ | $X$ | $O$ | $DRAW$ |
   |---|---|---|---|
   | 90 | 6012833 | 2893679 | 1093487 |
   | 3 | 6018914 | 2889949 | 1091136 |
   | 42 | 6015503 | 2892290 | 1092206 |
   | 10 | 6017533 | 2891538 | 1090928 |
   | 12 | 6019613 | 2891663 | 1088723 |

   From my MC Simulation results, I've deduced that Player X will win 60.17% of the games, Player O will win 28.92% of the games, and there will be a draw in 10.91% of the games.

   (b) **Explain your interpretation of playing "at random".**
   My interpretation of "at random" is two people (two sources/streams of randomness) randomly choosing spots on the board to mark. There are 9 spots to choose from, which corresponds to an

array of length 9. So for each player, they make an Equilikely choice from 0 to 8. They can mark their choice only if their selected spot is blank, and unmarked by themselves or the other player. If the spot they chose is marked, they continue making Equilikely choices until they choose an unmarked spot.

(c) **Compute the axiomatic probabilities and compare with your simulation results.** The axiomatic probabilities are 60.2%, 28.9%, and 10.9%. My simulation results were very close.

CODE FOR STICK BREAK FOLLOWS :

```
#include <stdio.h>
#include "rngs.h"

#define N 9999999                              /* number of replications */

//code for the random stick break problem.
//determine the odds that the three pieces can form a triangle
double Uniform(double a, double b)
/* ------------------------------------------
* generate a Uniform random variate, use a < b
* ------------------------------------------
*/
{
return (a + (b - a) * Random());
}
double breakStick(void)
{
SelectStream(0);
return Uniform(0.0,1.0);
}
int main(int argc, char const *argv[])
{
long   i;                                   /* replication index      */
double point_a, point_b, len1, len2, len3;
long   count[3] = {0};                     /* histogram used to keep track of whet
PlantSeeds(0);
for(i=0;i<N;i++)
{
//uniformily choose two break points from 0 to 1
point_a = breakStick();
point_b =  breakStick();
//calculate the lengths of the pieces based on our two break points
if(point_b >= point_a)
{
len3 = point_b - point_a;//middle piece length
```

```
len2 = 1- point_b;//rightmost piece length
len1 = point_a;//leftmost piece length
}else{
len3 = point_a - point_b;
len2 = 1 - point_a;
len1 = point_b;

}
/*for any triangle, the sum of the lengths of any two sides
must be greater than or equal to the length of the remaining side*/
if(((len1+len2) >= len3) && (len2+len3 >= len1) && (len3+len1 >= len2))
{
count[1]++;
}else{
count[2]++;
}
}
//printf("testing uniform breaking... %f ... %f...\n", Uniform(0.0,1.0), Unifo
printf("we have a triangle %ld times out of %ld times\n",count[1], N);
printf("we have a triangle %lf percent of the time\n", 100* (double) count[1]/N
return 0;
}
```

CODE FOR TIC TAC TOE FOLLOWS:

```
#include <stdio.h>
#include "rngs.h"
#define N 9999999
//code for testing probability when playing Tic Tac Toe on a 3x3 board at rando
//Gives probabilities on whether player 1 wins, player 2 wins, or there is a dr
//Player 1 always goes first
//Both players represent sources of randomness
//at random : always chooses a random number between 0 and 9. If their choice i
int Equilikely(long a, long b)
{
//Make an Equilikely choice on the game board from spots 0 to 9
```

```
return (a + (long) ((b - a + 1) * Random()));
}
int main() {

long win_count[2],z;
win_count[0]=win_count[1]=0;//each player starts with no wins
printf("Player 1 starts with %ld wins\n",win_count[0]);//testing
printf("Player 2 starts with %ld wins\n",win_count[1]);//testing
PlantSeeds(0);
for(z = 0; z < N; z++)
{
int index;
int sign;
int player;
int data[9] = {0};//create blank board
int win = 0;
int i = 0;
int count = 0;

while (count < 9 && win == 0) //while there are still moves to make and nobody
{
if(count % 2 == 0)//0,2,4,6,8 belong to player 1
{
sign = 'X';//player 1's symbol
player = 1;
SelectStream(0); //player 1's source of randomness

}else{//all other move numbers belong to player 2
sign = 'Y';//player 2's symbol
player = 2;
SelectStream(1);//player 2's source of randomness
}
//make an Equilikely choice from 1 to 9 to simulate random move
index = Equilikely(0, 8);
//if the selected spot has already been chosen, choose again.
while (data[index] != 0)
{
index = Equilikely(0, 8);
}

data[index] = sign;//make the blank spot belong to current player
//printf("player %d has chosen spot %d\n", player, index);
//Win checking : diagonal line
```

```
if((data[0]!= 0 && data[0] == data[4] && data[0] == data[8]) ||
(data[2]!= 0 && data[2] == data[4] && data[0] == data[6]))
{
win = 1;
win_count[player-1]++;
}



//win checking : rows
for(i = 0; i<7; i+=3)
{
if( data[i]!=0 && data[i] == data[i+1] && data[i] == data[i+2])
{
win_count[player-1]++;
win = 1;
}
}
//win checking : columns
for(i = 0; i <= 2; i++)
{
if(data[i]!=0 && data[i] == data[i+3] && data[i] == data[i+6])
{
win_count[player-1]++;
win = 1;
}

}
count++;//on to the next move
}
}

long draws = (N - (win_count[0]+win_count[1]));

printf("PLAYER 1 has won %ld times out of %ld random move games\n",win_count[0]
printf("PLAYER 2 has won %ld times out of %ld random move games\n",win_count[1]
printf("There has been a DRAW %ld times out of %ld random move games\n", draws,

printf("\nPLAYER 1 has won %lf percent of the games\n",100 * (double) (win_coun
printf("PLAYER 2 has won %lf percent of the games\n", 100 * (double) win_count
printf("%lf percent of the games have ended in DRAWS\n",100 * (double) draws/N
```

```
}
```