Pseudokod

# Omega

Omega(n, m) // n = Number of throws, m = number of die sides

omega[][] // We create a two-dimensional list

list[] // We create a simple list

permutations = // The total number of permutations

// for each loop, create a vector

for (i = 0; i < permutations; i++)

// for each loop, fill a vector

for (j = n - 1; j >= 0; j--)

// The new element is obtained using the formula below.

// The total number of possible outcomes divided by the number

// of die m to the power of whatever inner loop we are in - modulo  
// the number of sides of the die. This will iterate through the list.

newElement = (

Put newElement in list[].

// Put the vector in the vector-vector-reference

Put list[] in omega[][].

Function Omega for parameters n and m, where n represents the number of die throws and m represents the number of die sides.

Create a vector of vectors of integers, the sample space.

Create a vector of integers named list.

The total number of possible permutations is m to the power of n (or)

for each i less than the total number of permutations:

for each j greater than or equal to 0; j is decremented by 1, loop:

The new element is equal to i divided by modulo m + 1 (or) (

Put the new element in the vector of integers, list.

Put the list in the vector of vectors of integers, the sample space.

Print the sample space.

# Omega

integer n tärningskast

integer m sidor

integer[][] Omega( n, m )

{

integer[][] omega

integer i, j

for( 0 < i < m^n)

{

for( 0 < j < n )

{

omega[i][j] = (i/( m^j ) %m )+1

}

}

return omega

}

# E

integer[][] E( n, m )

{

integer[][] e

integer i, die

boolean allValues, set to True

for( 0 < i < omega.size )

{

for( 1 < die <= m )

{

if ( omega[i] does not contain die )

{

allValues = false

}

}

if( allValues = true )

{

put omega[i] in e

}

}

return e

}

# F

integer[][] F( n, m )

{

integer[][] f

integer i, j

boolean lessThan, set to False

for ( 0 < i < omega.size )

{

for ( 1 < j < omega[i].size )

{

if ( omega[i][j] < omega[i][j-1] )

{

lessThan = true

}

}

if ( lessThan = false )

{

put omega[i] in f

}

}

return f

}

# Section

integer[][] A, B

integer[][] section( A, B )

{

integer[][] sect

integer i, j

for ( 0 < i < A.size )

{

for ( 0 < j < B.size )

{

if ( A[i] == B[i] )

{

put A[i] in sect

}

}

}

return sect

}

# Probability

Probability()

{

float pE = ( e.size / omega.size )

float pF = ( f.size / omega.size )

float pE|F = ( sect.size / f.size )

print “P(E):” pE

print “P(F):” pF

print “P(E|F):” pEF

}

# Kasta\_tarning

integer kasta\_tarning( m )

{

if ( m < 2 )

{

return NULL

}

integer value = randomValue % m

return value

}

# Kasta\_tarningar

integer n, m

integer[] kasta\_tarningar( n, m )

{

integer[] values

for ( 0 < i < n )

{

put kasta\_tarning( m ) in values

}

return values

}

# OneBProbability

Integer n, m, x

oneBprobability( n, m, x )

{

Integer[][] values;

for ( 0 < i < x )

{

vector<int> tempValues = kasta\_tarningar(n, m);

values.push\_back(tempValues);

}

omega = values;

cout << "E: " << E(n, m).size() << endl;

int b = F(n, m).size();

cout << "F: " << b << endl;

int a = section(E(n, m), F(n, m)).size();

if (b != 0)

{

float probability = (float) a / (float) b;

cout << a << endl;

cout << b << endl;

cout << n << " " << m << "-sided dice " << "thrown " << x << " times\n gives P(E|F): " << probability << endl;

}

else

{

cout << a << endl;

cout << b << endl;

cout << "Don't divide by zero, stupid." << endl;

}

}