Laboratory 5 - Random Algorithms

I. Random number generation

In R there is a large collection of functions dedicated to generating random objects (like numbers, permutation etc).

Generating random integers. For generating integer random numbers one can use the *sample*() function; for example if you want to generate a random integer number between 1 and 300 or five random integer numbers between 200 and 250:

```
> x = sample(300,1)
> x
> [1] 68
> x = sample(200:250,5)
> x
> [1] 182 36 234 204 170 286
```

We can generate integer numbers with repetition (by default *replace* is FALSE):

```
> x = sample(30, 6, replace=T)
> x
> [1] 8 25 25 29 8 2
> x = sample(20:40, 5, replace=T)
> x
> [1] 20 30 37 24 30
```

The function *sample*() can sample from elements of a specified vector:

```
> x = c(2.1, 3.2, 2.3, 2.5, 3.1, 2.9, 2.6, 2.2, 3.3)
> sample(x, 5)
> [1] 3.1 2.2 3.2 2.9 3.2
> sample(x, 5, replace=T)
> [1] 3.1 2.2 3.2 2.9 3.2
```

We can use it also for generating random permutations:

```
> sample(10)
> [1] 3 2 5 7 8 10 6 9 1 4
> [sample(15)
> [1] 9 6 7 4 11 15 10 3 12 2 1 13 8 5 14
```

You are encouraged to take a look at other similar functions like *shuffle() from the permute package.*

Generating random real numbers. For generating random real numbers belonging to a given interval we can use *runif()* function; for example if you want to generate k uniform random numbers between a and b we can use *runif(k, a, b)*:

```
> runif(10,2, 4.5) # ten uniform random values between 2 and 4.5
> [1] 3.802909 3.072721 3.615275 2.378011 3.281129 4.269154
> [7] 2.611120 4.297596 2.418020 2.536075
> x = runif(4, 0, 1) # four uniform random values between 0 and 1
> x
> [1] 0.9979809 0.6081421 0.4032731 0.8214655
```

II. Random algorithms.

Solved exercise. (Monte Carlo) Write a function that implements the randomized verifying matrix multiplication algorithm. The function will have as parameters the matrices A, B, and C. Using this function write another function that uses the amplification method and reduces the probability error below 2^{-k}.

Solution. The function *matrix(data, nrow, ncol, byrow)* creates a matrix (*data* is a list of elements that will fill the matrix and *byrow* is FALSE by default):

```
> x = c(1, 3, 1, 4, 12, 7)
> M = matrix(x, 3, 2)# creates a matrix 3x2
> N= matrix(x, 2, 3)# creates a matrix 2x3
```

The following function verifies if $ABr = Cr$ for uniformly random generated vector r with components in $\{0, 1\}$.

```
matrix_product = function(A, B, C) {
  n = nrow(A);
  r = matrix( , nrow = n, ncol = 1);
  x = matrix( , nrow = n, ncol = 1);
  y = matrix( , nrow = n, ncol = 1);
  r = sample(0:1, n, replace = TRUE);
  for(i in 1:nrow(B)) {# x = Br
    x[i] = 0;
    for(j in 1:n)
      x[i] = (x[i]+ B[i,j]*r[j])%%2;
  }
  for(i in 1:n) {# y = Ax = ABr
    y[i] = 0;
    for(j in 1:nrow(B))
      y[i] = (y[i]+ A[i,j]*x[j])%%2;
  }
  for(i in 1:n) {# x = Cr
    x[i] = 0;
    for(j in 1:n)
      x[i] = (x[i]+ C[i,j]*r[j])%%2;
  }
  for(i in 1:n) {# verify if ABr==Cr
    if(y[i] !=x[i])
      return(FALSE);
  }
  return(TRUE);
}
```

The following fonction gives an error probability at most 2^{-k}.

```
matrix_product_reduce = function(A, B, C, k) {
  for(i in 1:k)
    if(!matrix_product(A, B, C))
      return(FALSE);
  }
  return(TRUE);
}
```

Solved exercise. (Las Vegas.) Write a function that evaluates a game tree of depth $2h$; the tree is given as a vector of 4^h values of its leaves.

Solution. We consider the input data as an array of 4^h bits like this (for $h = 2$)

```
> leaves = c(0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0)
```

It is easy to observe that an MAX node belongs to an odd level and a MIN node to an even level. Therefore a node i and must satisfy

$$[\log_2(i)] \equiv \begin{cases} 1\,(mod\,2), & \text{if } i \text{ is MAX} \\ 0\,(mod\,2), & \text{if } i \text{ is MIN} \end{cases}$$

The nodes are numbered starting from the root (node 1) to the leaves and from left to right. We evaluate all the nodes except the leaves; a leaf or a parent of a leaf, i, is characterized by the following inequality $\log_2(i) \geqslant (2h-1)$.

The following function recursively evaluates a given node i which is not a leaf (the function must be completed with an *if* corresponding to MAX nodes):

```
tree_eval = function(i, leaves) {
  a = runif(1, 0, 1); len = length(leaves);
  if(log(i,2) >= log(len,2) - 1) { # the children of node i are leaves
    if(a <= 0.5) {
      if(leaves[2*i - len + 1] == 0)
        return(leaves[2*i +1 -len + 1];
      return(1);}
    else {
      if(leaves[2*i + 1 -len + 1] == 0)
        return(leaves[2*i - len + 1];
      return(1);}
  }
  if((floor(log(i,2))%% 2 == 0)){ # the node i is a MIN one
    if(a <= 0.5) {
      if(tree_eval (2*i, leaves) == 1)
        return(tree_eval (2*i + 1, leaves));
      return(0);
    else {
      if(tree_eval (2*i +1, leaves) == 1)
        return(tree_eval (2*i, leaves));
      return(0);
  }
  ......
}
```

After complete the description of the above function you can use it like follows

```
game_tree_eval = function(leaves) {
  return(tree_eval (1, leaves));
}
```

Verify the result computing by hand the value of the tree.

Exercises to work.

1. Write a function that simulates the following finite random variable

$$X : \begin{pmatrix} x_1 & x_2 & \dots & x_k \\ p_1 & p_2 & \dots & p_k \end{pmatrix}$$