3.1 Random Surfer - Markov Chain Monte Carlo

This exercise largely corresponds to the "Random Surfer" (or Journey of the ant) slides from the lectures.

Your task is to complement the skeleton code from http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex03-1/skeleton/Main.java, eventually implementing a random walk through both simulation and iterative computation. Note that the code compiles even without filling any gaps. You can test individual parts separately and submit partial solutions to the judge to get the individual parts evaluated. Note that there may be dependencies: please process the tasks in the provided order.

We provide sample input data to the test program `Main` at the end of the source code. Use them in order to test your code locally before submitting to the judge. The judge uses different input data for evaluating your code.

3.1.1 Matrix Input

In this exercise the goal is to read in a $r \times c$ matrix from the input. Implement the following method

```
// pre: matrix data provided via scanner
// post: returns matrix data
public static double[][] readMatrix(java.util.Scanner scanner)
```

in order to read a matrix from the input using the Scanner. The format of the input data is explained with the following 2×3 example matrix:

```
2 3
0.10000 0.20000 0.30000
1.10000 1.20000 1.30000
```

The first line contains the number r of rows followed by the number c of columns, both integers. The following r lines contain c doubles each, representing a $r \times c$ matrix. The numbers are separated by whitespaces.

Validate your solution here: https://challenge.inf.ethz.ch/team/websubmit.php?problem=IB16032

3.1.2 Vector-Matrix-Multiplication

In this exercise the goal is to do a vector-matrix multiplication ($v * m$). Implement the following method

```
// pre: non-empty input vector of length N, matrix of size N times M
// post: return vector-matrix product v * m in a vector of size M
public static double[] multiplyVectorMatrix(double[] v, double[][] m)
```

to compute the product of a vector of length N and a matrix of size $N \times M$, $N, M > 0$. Return a new vector with the result. Do not modify the input data.

Validate your solution here: https://challenge.inf.ethz.ch/team/websubmit.php?problem=IB16033

3.1.3 Simulation

In this exercise you simulate the behavior of the random surfer. The random surfer starts in state 0 and makes t steps according to the transition matrix P. The result is a vector containing the probabilities that the surfer is on a specific page after step t. Implement the following method

```
// pre: non-empty probability matrix P, t >= 0 number of steps
// post: return relative frequencies of visits in each point
//       after MCMC simulation until convergence reached
public static double[] simulate(int t, double[][] P)
```

to simulate the random surfer. You should use the already existing method `simulateLine` to simulate a single step (transition to the next page), just as we did in the lectures.

Validate your solution here: https://challenge.inf.ethz.ch/team/websubmit.php?problem=IB16034

3.1.4 Convergence (Optional)

This exercise is optional, but we encourage you to make the following experiment. Use the transition matrix m =

```
0.10000 0.30000 0.20000 0.10000 0.30000
0.10000 0.20000 0.10000 0.30000 0.30000
0.30000 0.10000 0.30000 0.20000 0.10000
0.50000 0.50000 0.00000 0.00000 0.00000
0.90000 0.00000 0.00000 0.10000 0.00000
```

and starting with $v = (1, 0, 0, 0, 0)$ compute $v = v \cdot m$ iteratively.
What do you observe for v when the number of iterations increase?
Repeat with a different start vector v. What do you observe?

For this experiment you can use the following method:

```
// pre: non-empty probability matrix P, i >= 0 number of iterations
// post: return vector-matrix product v * m after i iterations
public static double[] multipleMultiplications(int i, double[] v, double[][] m)
```

which executes the Vector-Matrix-Multiplication i times.

Compare the result with the outcome of the random surfer experiment for a large number of steps.

3.2 Introduction to Hash-Functions

For this exercise you should use the skeleton code from http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex03-2/skeleton/Main.java This exercise is an introduction to hashing. A hash function on a string is a method that aims at mapping a string s to an integer value $v(s)$ and tries to come up with different values $v(s)$ for different strings s. For instance with the hash function provided below the string *ETH* maps to the value 58.

In the following, we look at a particular example of a hash-function. The function iterates over a string s, similar to iterating over an array. Each character is multiplied by the factor b^i, where b is a prime number and i is the iteration count. The formula for this hash function looks like this:

$$sum = \sum_{i=0}^{s.length-1} (s[i] * b^{(i+1)})$$

After computing the sum, the hash value is computed by taking the modulo with respect to m which maps the value into the range $(0, m)$.

$$key = (sum \mod m)$$

The prime number $b = 31$ and the modulo $m = 101$ are already defined in the provided skeleton code.

Of course, by the limited size of the domain ($m = 101$), it is impossible to avoid that different strings get the same value. When different strings map to the same value, we call this a *collision*. For example *students* and *computer* collide when $b = 31$ and $m = 101$

Implement the described function in the following method

```
// pre: non-empty string s
// post: returns value representing the string s
public static int computeHash(String s)
```

Validate your solution here: https://challenge.inf.ethz.ch/team/websubmit.php?problem=IB16036

We encourage you to make some experiments with different choices of m and b in order to develop a feeling how this works.