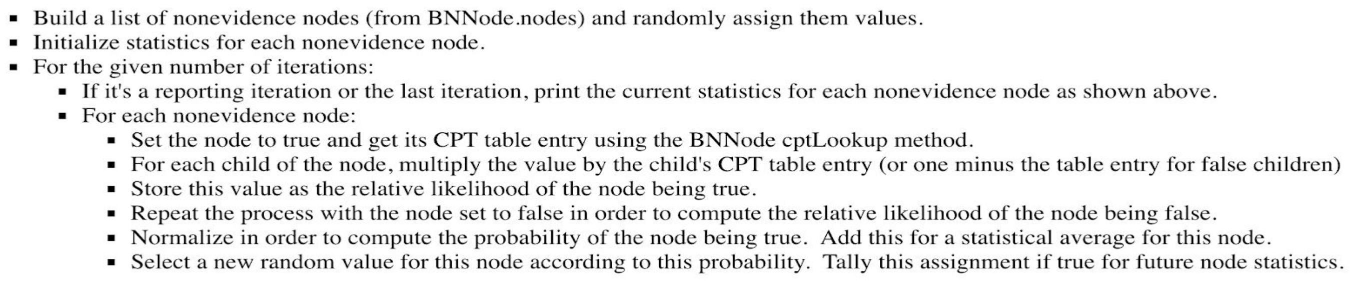
CSE 150 Final Project
Gibbs Sampling

Team member: Xinyu Li, John Jun, Ziwei Chen

Project description:

In this part of the project, our group is aiming to (1) implement **simulate()** method in **BNGibbsSampler.java** and (2) apply **simulate()** method on Exercises 2, 4, 5, 6, 7 and 8. What this **simulate()** function does is it will calculate the average conditional probability and the fraction of iterations the variable has been assigned true. This method can be compared to the alternative technique using the EM CPT update rule. By using this different approach, we attempt to inspect the result and look for patterns (i.e. convergence of probability, experimental demonstration of markov blanket, inference relationships).

- The algorithm to implement **simulate()** method is given below:



- Build a list of nonevidence nodes (from BNNode.nodes) and randomly assign them values.
- Initialize statistics for each nonevidence node.
- For the given number of iterations:
    - If it's a reporting iteration or the last iteration, print the current statistics for each nonevidence node as shown above.
    - For each nonevidence node:
        - Set the node to true and get its CPT table entry using the BNNode cptLookup method.
        - For each child of the node, multiply the value by the child's CPT table entry (or one minus the table entry for false children)
        - Store this value as the relative likelihood of the node being true.
        - Repeat the process with the node set to false in order to compute the relative likelihood of the node being false.
        - Normalize in order to compute the probability of the node being true. Add this for a statistical average for this node.
        - Select a new random value for this node according to this probability. Tally this assignment if true for future node statistics.

<Figure 1> Gibbs sampling algorithm
(retrieved from *http://modelai.gettysburg.edu/2017/mc2/index.html*)

The sample output is given below:

```
BNNode a:
  value: false
  parents: (none)
  children: b c
  CPT: 0.2
BNNode b:
  value: false
  parents: a
  children: d
  CPT: 0.2 0.8
BNNode c:
  value: false
  parents: a
  children: d e
  CPT: 0.05 0.2
BNNode d:
  EVIDENCE
  value: false
  parents: b c
  children: (none)
  CPT: 0.05 0.8 0.8 0.8
BNNode e:
  EVIDENCE
  value: true
  parents: c
  children: (none)
  CPT: 0.6 0.8
```

```
After iteration 200000:
Variable, Average Conditional Probability, Fraction True
a, 0.0971144285711284, 0.096385
b, 0.09707124999969938, 0.09655
c, 0.031003729284424134, 0.031815
```

```
After iteration 400000:
Variable, Average Conditional Probability, Fraction True
a, 0.09726980357149692, 0.096775
b, 0.09716282142863672, 0.0970625
c, 0.03109983597229418, 0.0315075
```

```
After iteration 600000:
Variable, Average Conditional Probability, Fraction True
a, 0.09744563095330394, 0.09715
b, 0.09732005952473075, 0.097475
c, 0.031158255337808277, 0.03146
```

```
After iteration 800000:
Variable, Average Conditional Probability, Fraction True
a, 0.09743473214411201, 0.09719375
b, 0.09732475000125447, 0.0974875
c, 0.031153070745652163, 0.03141625
```

```
After iteration 1000000:
Variable, Average Conditional Probability, Fraction True
a, 0.09725659285855631, 0.096923
b, 0.09716166428712687, 0.097158
c, 0.031087865742980978, 0.031275
```

<Figure 2> Sample output of Gibbs Sampling with input file **pearl.in**

- We can see that the average conditional probability and fraction true converges after a large number of iterations. This is in part due to minimization of sampling errors through a large number of iterations.

- We also would like to reflect on reasons for using Gibbs sampling instead of the variable elimination technique;
    - "Variable Elimination is a fairly efficient way to perform *exact* inference (i.e., to compute exact probabilities) on models where the DAG is a polytree. Gibbs sampling is an alternative that runs more efficiently for large models, but produces only approximate results". (From piazza post @383: https://piazza.com/class/jm99ybdug904de?cid=383)

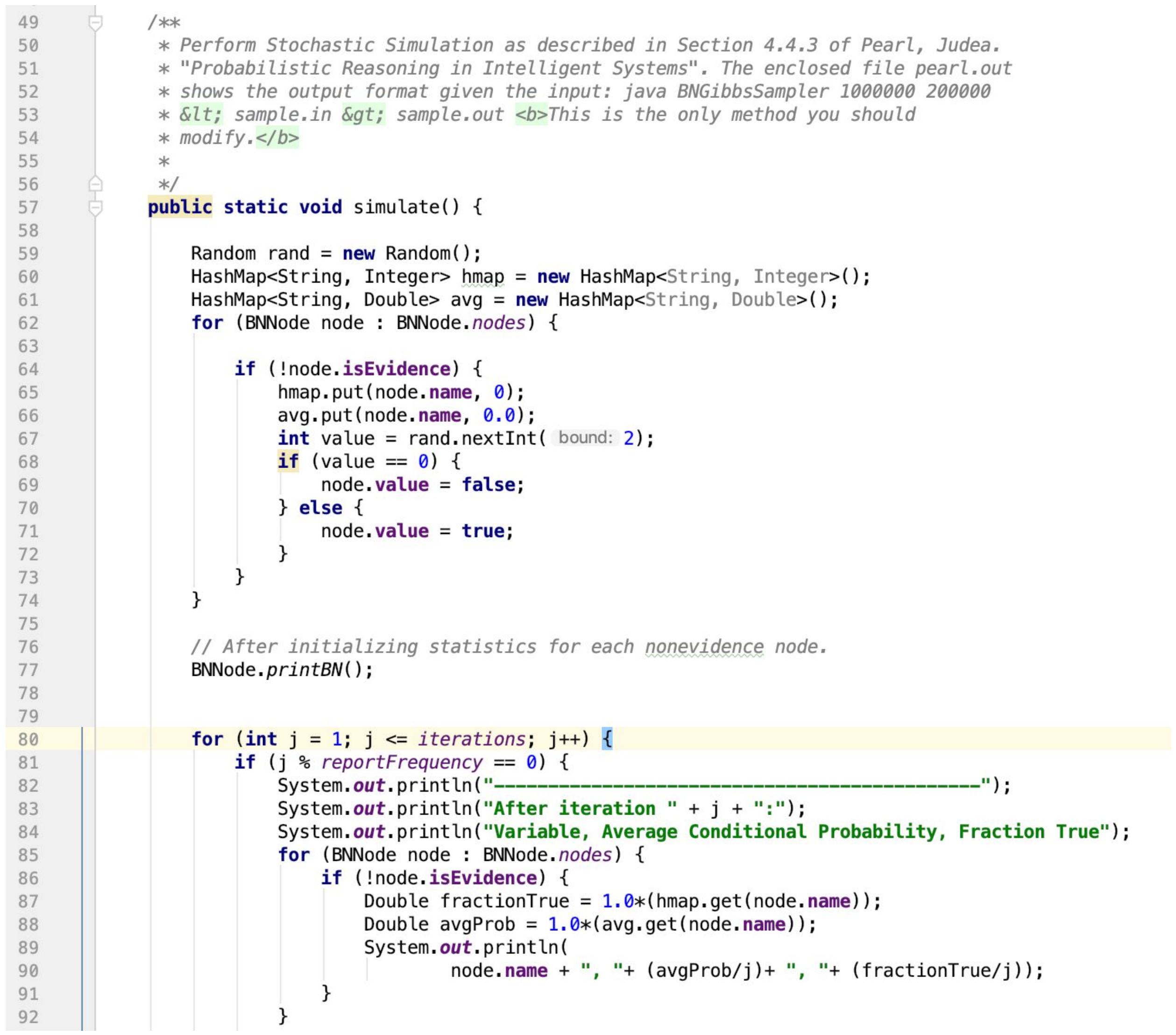
- Finally, we would like to extend the application of the simulate() method to solve cases presented in Exercises 2, 4, 5, 6, 7 and 8. These questions deal with different types of inference techniques/types and can be answered by modifying the input file for the program.

<u>Exercise 1</u>

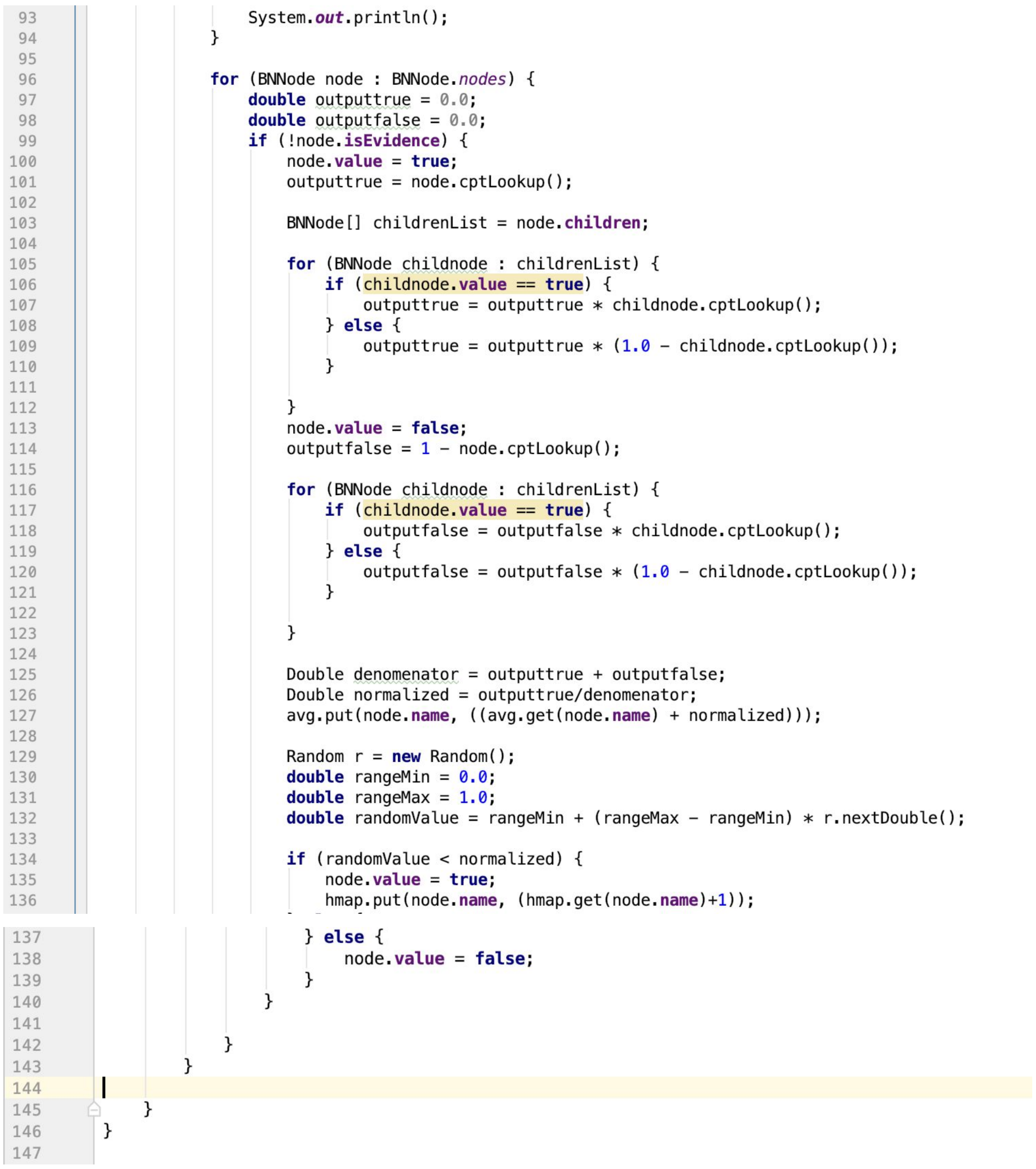
Stochastic Simulation with Gibbs Sampling: Implement Gibbs sampling of a Bayesian network as described in Section 4.4.3 of "Probabilistic Reasoning in Intelligent Systems" by Judea Pearl. Much convenient groundwork has been done for you. In file modelai-mcmc-dist.zip, you will find (among other things) an Eclipse project containing a collection of Java files that parse input and construct a Bayesian network data structure for you. You should only be concerned with implementing the simulate() method in BNGibbsSampler.java. Do not modify other parts of the code.

Implementation

I



```java
/**
 * Perform Stochastic Simulation as described in Section 4.4.3 of Pearl, Judea.
 * "Probabilistic Reasoning in Intelligent Systems". The enclosed file pearl.out
 * shows the output format given the input: java BNGibbsSampler 1000000 200000
 * &lt; sample.in &gt; sample.out <b>This is the only method you should
 * modify.</b>
 *
 */
public static void simulate() {

    Random rand = new Random();
    HashMap<String, Integer> hmap = new HashMap<String, Integer>();
    HashMap<String, Double> avg = new HashMap<String, Double>();
    for (BNNode node : BNNode.nodes) {

        if (!node.isEvidence) {
            hmap.put(node.name, 0);
            avg.put(node.name, 0.0);
            int value = rand.nextInt( bound: 2);
            if (value == 0) {
                node.value = false;
            } else {
                node.value = true;
            }
        }
    }

    // After initializing statistics for each nonevidence node.
    BNNode.printBN();


    for (int j = 1; j <= iterations; j++) {
        if (j % reportFrequency == 0) {
            System.out.println("---------------------------------------------");
            System.out.println("After iteration " + j + ":");
            System.out.println("Variable, Average Conditional Probability, Fraction True");
            for (BNNode node : BNNode.nodes) {
                if (!node.isEvidence) {
                    Double fractionTrue = 1.0*(hmap.get(node.name));
                    Double avgProb = 1.0*(avg.get(node.name));
                    System.out.println(
                            node.name + ", "+ (avgProb/j)+ ", "+ (fractionTrue/j));
                }
            }
```



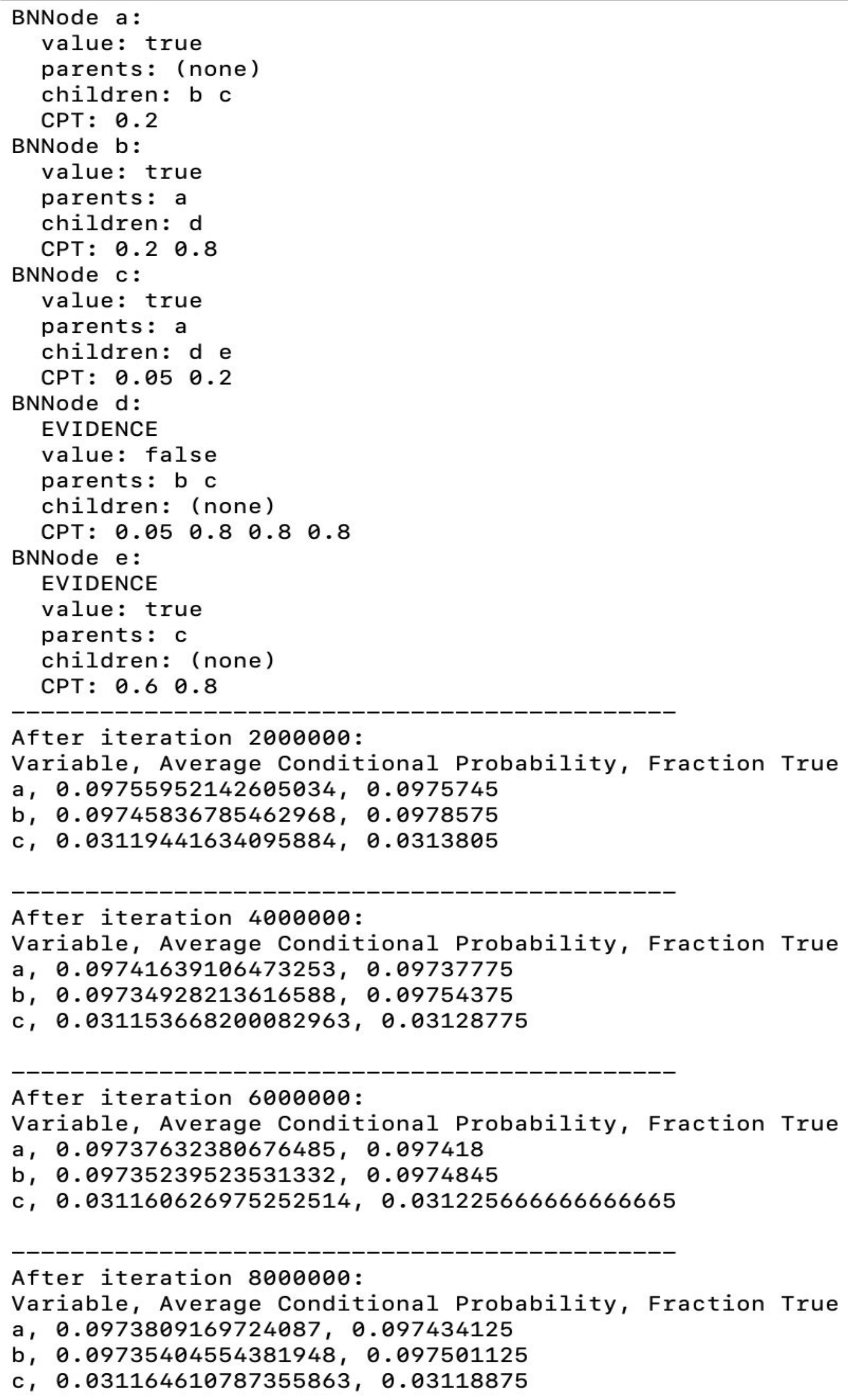
```
                System.out.println();
            }

            for (BNNode node : BNNode.nodes) {
                double outputtrue = 0.0;
                double outputfalse = 0.0;
                if (!node.isEvidence) {
                    node.value = true;
                    outputtrue = node.cptLookup();

                    BNNode[] childrenList = node.children;

                    for (BNNode childnode : childrenList) {
                        if (childnode.value == true) {
                            outputtrue = outputtrue * childnode.cptLookup();
                        } else {
                            outputtrue = outputtrue * (1.0 - childnode.cptLookup());
                        }

                    }
                    node.value = false;
                    outputfalse = 1 - node.cptLookup();

                    for (BNNode childnode : childrenList) {
                        if (childnode.value == true) {
                            outputfalse = outputfalse * childnode.cptLookup();
                        } else {
                            outputfalse = outputfalse * (1.0 - childnode.cptLookup());
                        }

                    }

                    Double denomenator = outputtrue + outputfalse;
                    Double normalized = outputtrue/denomenator;
                    avg.put(node.name, ((avg.get(node.name) + normalized)));

                    Random r = new Random();
                    double rangeMin = 0.0;
                    double rangeMax = 1.0;
                    double randomValue = rangeMin + (rangeMax - rangeMin) * r.nextDouble();

                    if (randomValue < normalized) {
                        node.value = true;
                        hmap.put(node.name, (hmap.get(node.name)+1));
                    } else {
                        node.value = false;
                    }
                }

            }
        }

    }
}
```

<Figure 3> Implementation of **simulate()** method in Java

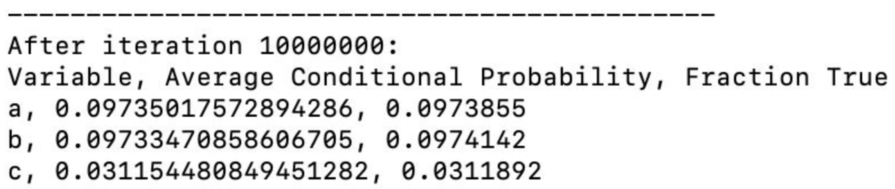
- All the resources involved (started code, data sets) are contained in the webpage provided: http://modelai.gettysburg.edu/2017/mc2/index.html. All necessary files will be downloaded directly from the website.

Comparing our outputs with the sample output (for verification)

- Our results is given below:



```
BNNode a:
  value: true
  parents: (none)
  children: b c
  CPT: 0.2
BNNode b:
  value: true
  parents: a
  children: d
  CPT: 0.2 0.8
BNNode c:
  value: true
  parents: a
  children: d e
  CPT: 0.05 0.2
BNNode d:
  EVIDENCE
  value: false
  parents: b c
  children: (none)
  CPT: 0.05 0.8 0.8 0.8
BNNode e:
  EVIDENCE
  value: true
  parents: c
  children: (none)
  CPT: 0.6 0.8
---------------------------------------------
After iteration 2000000:
Variable, Average Conditional Probability, Fraction True
a, 0.09755952142605034, 0.0975745
b, 0.09745836785462968, 0.0978575
c, 0.03119441634095884, 0.0313805

---------------------------------------------
After iteration 4000000:
Variable, Average Conditional Probability, Fraction True
a, 0.09741639106473253, 0.09737775
b, 0.09734928213616588, 0.09754375
c, 0.031153668200082963, 0.03128775

---------------------------------------------
After iteration 6000000:
Variable, Average Conditional Probability, Fraction True
a, 0.09737632380676485, 0.097418
b, 0.09735239523531332, 0.0974845
c, 0.031160626975252514, 0.031225666666666665

---------------------------------------------
After iteration 8000000:
Variable, Average Conditional Probability, Fraction True
a, 0.0973809169724087, 0.097434125
b, 0.09735404554381948, 0.097501125
c, 0.031164610787355863, 0.03118875
```



```
--------------------------------------------
After iteration 10000000:
Variable, Average Conditional Probability, Fraction True
a, 0.09735017572894286, 0.0973855
b, 0.09733470858606705, 0.0974142
c, 0.031154480849451282, 0.0311892
```

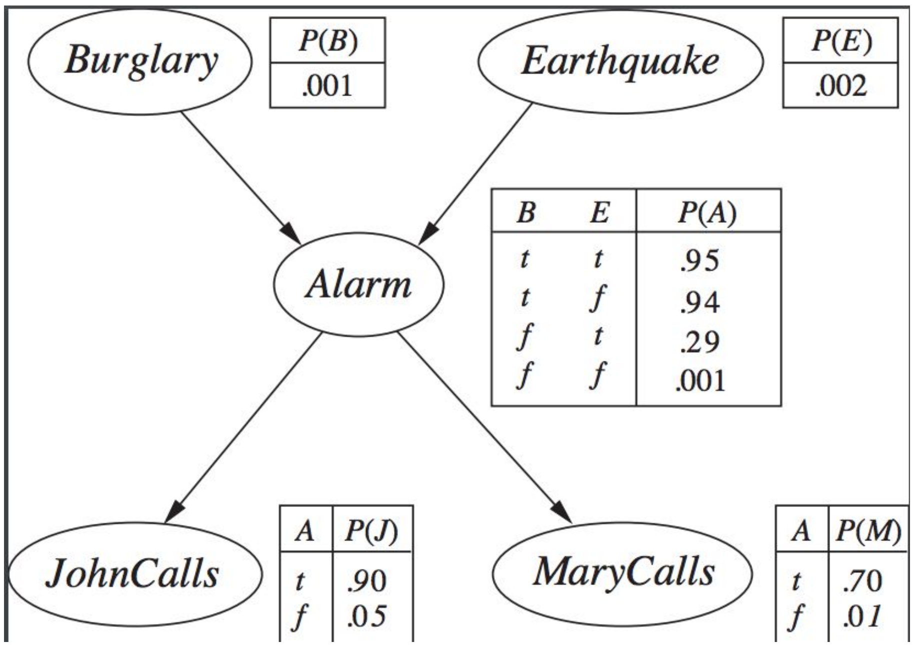
<Figure 4> Output of the Implemented **simulate()** method

- Our results is almost identical to the sample output. The negligible difference can be explained by the following two reasons:
    - We are keeping a different number of decimal places than the sample code. This could cause differences due to rounding issues.
    - In the step of assigning a random value of a current node based on the normalization of true value, we might be handling a different assignment on the edge case, i.e., when the random value is equal to the normalized. The sample output may be a consequence of assigning the node to be true when equality occurs, while we assign the node to be false in such circumstances.

Based on the comparison of our output to the example output and understanding that the minute differences in the values are due to comparison type difference and/or rounding decimal place discrepancy.

Exercise 2

Representation and reasoning: Use your BNGibbsSampler.java implementation for this and the following questions. Encode the Bayesian network of Russell & Norvig Figure 14.2 for input to the program above as file alarm.in. What are the probabilities of nonevidence variables with no evidence? With Mary calling? With Mary calling and positive evidence of an earthquake?



P(B)
.001

P(E)
.002

B	E	P(A)
t	t	.95
t	f	.94
f	t	.29
f	f	.001

A	P(J)
t	.90
f	.05

A	P(M)
t	.70
f	.01

1. Probabilities of nonevidence variables with no evidence:

```
After iteration 10000000:
Variable, Average Conditional Probability, Fraction True
b, 9.884177369766187E-4, 9.856E-4
e, 0.001997494006839992, 0.0019987
a, 0.0025068996949939868, 0.0024966
j, 0.052122104991849605, 0.0520959
m, 0.011722652998068177, 0.0117244
```

<u>alarm.in</u>

```
P(b) = {.001}
P(e) = {.002}
P(a|b,e) = {.001, .29, .94, .95}
P(j|a) = {.05, .90}
P(m|a) = {.01, .70}
Evidence
```

2. Probabilities of nonevidence variables with Mary calling:

```
After iteration 10000000:
Variable, Average Conditional Probability, Fraction True
b, 0.05615575627084754, 0.0561579
e, 0.035879814848554924, 0.0358481
a, 0.15011947952941768, 0.1501259
j, 0.17760701002156787, 0.1775891
```

alarm.in

```
P(b) = {.001}
P(e) = {.002}
P(a|b,e) = {.001, .29, .94, .95}
P(j|a) = {.05, .90}
P(m|a) = {.01, .70}
Evidence
m
```

3. Probabilities of nonevidence variables with Mary calling and positive evidence of an earthquake:

```
After iteration 10000000:
Variable, Average Conditional Probability, Fraction True
b, 0.0031605511396931066, 0.003179
a, 0.9663202891576785, 0.9662682
j, 0.8713279651527617, 0.8714029
```

alarm.in

```
P(b) = {.001}
P(e) = {.002}
P(a|b,e) = {.001, .29, .94, .95}
P(j|a) = {.05, .90}
P(m|a) = {.01, .70}
Evidence
m
e
```

Exercise 4

Diagnostic inference concerns querying a nonevidence variable that is an ancestor of an evidence variable, reasoning backward from effects to causes. How does the introduction of true evidence of d affect the probability of a compared to our **pearl.in** network with no evidence?

- With introduction of true evidence d:
  The average conditional probability of node a converges to 0.42494375792307426

  ```
  After iteration 10000000:
  Variable, Average Conditional Probability, Fraction True
  a, 0.42494375792307426, 0.4248991
  b, 0.7999495568573438, 0.7998723
  c, 0.2000129101685073, 0.2000729
  e, 0.6400145199088393, 0.6399669
  ```

- With no evidence:
  The average conditional probability of node a converges to 0.20017099073304795

  ```
  After iteration 10000000:
  Variable, Average Conditional Probability, Fraction True
  a, 0.20017099073304795, 0.2000998
  b, 0.3202750778272954, 0.3203732
  c, 0.08004894159999792, 0.0800778
  d, 0.3203094449571781, 0.3203731
  e, 0.6160154999228837, 0.6159105
  ```

The introduction of true evidence d increases the probability of a being true.

Exercise 5

Causal inference concerns querying a nonevidence variable that is a descendant of an evidence variable, reasoning forward from causes to effects. How does the introduction of false evidence of a affect the probability of d compared to our pearl.in network with no evidence?

- With the introduction of false evidence of a:
  The average conditional probability of node d converges to 0.23020264501337848

  ```
  After iteration 10000000:
  Variable, Average Conditional Probability, Fraction True
  b, 0.20020092002898715, 0.2002452
  c, 0.05003331101745748, 0.0500314
  d, 0.23020264501337848, 0.2303244
  e, 0.6100062199268451, 0.6098096
  ```

- With no evidence:
  The average conditional probability of node d converges to 0.3203094449571781

  ```
  After iteration 10000000:
  Variable, Average Conditional Probability, Fraction True
  a, 0.20017099073304795, 0.2000998
  b, 0.3202750778272954, 0.3203732
  c, 0.08004894159999792, 0.0800778
  d, 0.3203094449571781, 0.3203731
  e, 0.6160154999228837, 0.6159105
  ```

The introduction of false evidence a decreases the probability of d being true.

Exercise 6

Intercausal inference (also called "explaining away") concerns a querying common cause variables of the same evidence effect variable. Suppose we have evidence only of a coma (d) in our pearl.in network. What are the probabilities of b and c? Suppose then that we learn that the patient has a brain tumor (c). What is the new probability of them having increased total serum calcium (b)? Suppose instead that we learn that the patient has increased total serum calcium (b)? What is the new probability of them having a brain tumor (c)?

- With introduction of true evidence d:
  The average conditional probability of node b converges to 0.7999495568573438
  The average conditional probability of node c converges to 0.2000129101685073

  ```
  After iteration 10000000:
  Variable, Average Conditional Probability, Fraction True
  a, 0.42494375792307426, 0.4248991
  b, 0.7999495568573438, 0.7998723
  c, 0.2000129101685073, 0.2000729
  e, 0.6400145199088393, 0.6399669
  ```

- With introduction of true evidence d and c:
  The average conditional probability of node b converges to 0.4998204600000534

  ```
  After iteration 10000000:
  Variable, Average Conditional Probability, Fraction True
  a, 0.49990362000000926, 0.4997008
  b, 0.4998204600000534, 0.4998393
  e, 0.7999999198711804, 0.7999915
  ```

- With introduction of true evidence d and b:
  The average conditional probability of node c converges to 0.12496374240587009

  ```
  After iteration 10000000:
  Variable, Average Conditional Probability, Fraction True
  a, 0.49999272003516265, 0.4998736
  c, 0.12496374240587009, 0.1249788
  e, 0.6249956999173516, 0.6248586
  ```

With evidence node d, if we know for sure we have either b or c as evidence, then the probability of the other node being true decreases.

<u>Exercise 7</u>

Mixed inference concerns a queried nonevidence variable that is both a descendant and an ancestor of evidence variables. For example, if the patient has no metastatic cancer (a) but is in a coma (d), what is the probability of the patient having a brain tumor (c) in our **pearl.in** network?

- With introduction of false evidence a and true evidence d (-a,d):
  The probability of the patient having a brain tumor © converges to 0.17414442770795258

```
After iteration 10000000:
Variable, Average Conditional Probability, Fraction True
b, 0.695398979921717, 0.6950771
c, 0.17414442770795258, 0.1743349
e, 0.6348669199116509, 0.6346552
```

Exercise 8

A weakness of Gibbs sampling: As with many Monte Carlo simulation algorithms, one must be careful when improbable (but not impossible) events are of great importance. In the Bayes network tree (tree.in) defined below, what are the true probabilities of each variable (assuming no evidence)? Perform Gibbs sampling with different orders of magnitudes of iterations (e.g. 10000, 100000, 1000000, 10000000), share your results, and reflect on what they teach.

```
BNNode a:
  value: false
  parents: (none)
  children: b c
  CPT: 0.1
BNNode b:
  value: true
  parents: a
  children: d e
  CPT: 0.0 0.1
BNNode c:
  value: true
  parents: a
  children: (none)
  CPT: 0.0 0.1
BNNode d:
  value: true
  parents: b
  children: f g
  CPT: 0.0 0.1
BNNode e:
  value: true
  parents: b
  children: (none)
  CPT: 0.0 0.1
BNNode f:
  value: false
  parents: d
  children: h i
  CPT: 0.0 0.1
```

```
BNNode g:
  value: false
  parents: d
  children: (none)
  CPT: 0.0 0.1
BNNode h:
  value: false
  parents: f
  children: j k
  CPT: 0.0 0.1
BNNode i:
  value: true
  parents: f
  children: (none)
  CPT: 0.0 0.1
BNNode j:
  value: true
  parents: h
  children: (none)
  CPT: 0.0 0.1
BNNode k:
  value: false
  parents: h
  children: (none)
  CPT: 0.0 0.1
--------------------------------------------
After iteration 2000000:
Variable, Average Conditional Probability, Fraction True
a, 0.10011417890294307, 0.100309
b, 0.010124596330273112, 0.0101035
c, 0.010030899999994278, 0.010008
d, 0.0010264357798162981, 0.0010465
e, 0.001010349999999629, 0.0010725
f, NaN, 1.055E-4
g, 1.046499999999962E-4, 1.165E-4
h, NaN, 7.0E-6
i, 1.0550000000000014E-5, 1.05E-5
j, 7.000000000000001E-7, 5.0E-7
k, 7.000000000000001E-7, 5.0E-7
```

```
-------------------------------------------
After iteration 4000000:
Variable, Average Conditional Probability, Fraction True
a, 0.10001098854058937, 0.100076
b, 0.01002966972471071, 0.010018
c, 0.010007599999924495, 0.009982
d, 0.0010063027522942635, 0.00100325
e, 0.0010017999999993627, 0.001021
f, NaN, 9.95E-5
g, 1.0032500000000569E-4, 1.0825E-4
h, NaN, 7.25E-6
i, 9.950000000000074E-6, 1.025E-5
j, 7.250000000000003E-7, 5.0E-7
k, 7.250000000000003E-7, 7.5E-7

-------------------------------------------
After iteration 6000000:
Variable, Average Conditional Probability, Fraction True
a, 0.09999524618019012, 0.10006633333333333
b, 0.01001098165129687, 0.0100075
c, 0.010006633333234467, 0.009971666666666667
d, 0.0010016896024460371, 9.956666666666666E-4
e, 0.0010007500000007187, 0.0010093333333333334
f, NaN, 9.483333333333334E-5
g, 9.956666666667788E-5, 1.0666666666666667E-4
h, NaN, 5.833333333333333E-6
i, 9.483333333333423E-6, 9.0E-6
j, 5.833333333333336E-7, 3.3333333333333335E-7
k, 5.833333333333336E-7, 5.0E-7

-------------------------------------------
After iteration 8000000:
Variable, Average Conditional Probability, Fraction True
a, 0.09994884288101975, 0.099998625
b, 0.009994496559544153, 0.009978
c, 0.009999862500021084, 0.0099535
d, 9.962339449530848E-4, 9.8925E-4
e, 9.978000000014386E-4, 0.001002375
```

```
f, NaN, 9.3875E-5
g, 9.892500000001392E-5, 1.045E-4
h, NaN, 6.25E-6
i, 9.387500000000001E-6, 9.0E-6
j, 6.249999999999997E-7, 2.5E-7
k, 6.249999999999997E-7, 7.5E-7

--------------------------------------------
After iteration 10000000:
Variable, Average Conditional Probability, Fraction True
a, 0.09995870732353772, 0.0999638
b, 0.010003249541189784, 0.0100016
c, 0.009996380000133077, 0.0099492
d, 9.976550458717557E-4, 9.986E-4
e, 0.0010001600000018854, 0.0010088
f, NaN, 9.38E-5
g, 9.986000000001585E-5, 1.039E-4
h, NaN, 6.8E-6
i, 9.379999999999895E-6, 9.0E-6
j, 6.799999999999992E-7, 3.0E-7
k, 6.799999999999992E-7, 7.0E-7
```

From this part, we can see that after a large number of iterations, the average conditional probability and the fraction true still don't converge. This serves to imply that Gibbs sampling does not have a good performance on a BN with no evidence node(s) defined.

Learning/Discussion/Conclusion

- Comparison between Variable Elimination and Gibbs sampling method

Method Type	Pros	Cons
Variable Elimination	Ouputs exact values	Not efficient on large models
Gibbs Sampling	Efficient on large models	- Give approximate values (will converge at the end). This is critical since the approximations are good when not asking for the exact values - Worse performance on BNs with no evidence node(s); see above (**Exercise 8**).

- The functionality of the **simulate()** method is similar to that of the EM algorithm in that both of them use iteration in order to obtain a converging result. However, the difference between Gibbs sampling and EM is that EM updates the probabilities using the parameter values and calculate expected count from the last iteration, and then update the probabilities use the count. On the other hand, Gibbs sampling does not update the CPT values; instead, we only keep track of the normalized true values and the average of those values. The normalized true values are used for random assignment of the node value based on this probability.

- ***Challenges***

    - The main challenge was to fully understand the concept behind Gibbs Sampling as the implementation of the iterative method was the core of the project. Unlike some of the practices we have done in class using EM update rules, which uses an iterative method to update the CPT, the Gibbs Sampling method fittingly *samples* the probabilistic outputs and aggregate the output information. Hence, in order to have a correct understanding of the concept and the detail of the algorithm, the team members conducted group meetings to discuss about the concept and to ensure that every member had a clear understanding to allow for active contribution to the project.

    - Once we had a running correct implementation of the simulate() method, which essentially iterates through a large number of times to sample and record the random output of each node based on their likelihood values of being true, we could simply generate a different input file to check how having a certain set of evidence and non-evidence nodes alter the results. The corresponding parts are Exercise 6, Exercise 7, and Exercise 8. The challenge here was to come up with a contextual interpretation of why such difference/similarity in the results occurred. In order to bring explanation, we employed the concept of markov blanket and the relationship between each node, dictated by the given probabilities in the CPT.

○ The optimal implementation was said to be around 60 lines of code. Therefore, we strived to implement the code using as few lines as possible. In order to keep track of the average conditional probability value, we used the **Hashmap** data structure since we need to be able to access the value immediately at the printing iterations. We made use of the **Random Number Generator** to perform random assignment of the values of the nodes based on their probability; when the randomly generated value is greater than the probability, we gave the node a false value and true value, otherwise. Since the **BNNode.java** file already had implementations of methods we could use to get information about individual nodes, we relied on them as much as possible. Hence, instead of having to keep track of the node values and CPT for calculation in each iteration, we utilized methods such as **cptLookup()** and used member variable accessing to immediately get children or the parents of a node and to get the value, and more. By utilizing the member variables and the already implemented methods, we could minimize the number of lines required to derive the same functionality.

○ Through the series of exercises, we have explored how to hardcode the implementation of an iterative Gibbs sampling method. Also, we have learned how different calculation methods may be applicable to a certain set of situations. Also, in order to work together as a group, we used git to do version control as we preferred to do remote work at our individual place of convenience.