Implement List

Part 1, Complete the construction of Bayesian network by using AISPACE tools. According to the description of conditions, points in the Bayesian network are generated, and the relationship between points is designed. Also, using the information to add the conditional probabilities for nodes that have parents, and the prior probabilities for nodes without parents. Based on the first Bayesian network, the construction of a second Bayesian network was completed. Compared to the first Bayesian network, the second network adds two more events and determines the relationship between the two new events and the existing nodes.

Part 2, Design and implement the BNT network generation algorithm. Through the guidance on the slides, the algorithm requirements are completed step by step. For example, get information from two networks and enter as input. Find internal and external points in the network and modify the dependencies between the points. The implementation of the algorithm relies on two networks provided by the teacher but does not use the Encog package.

The report contains totally 3454 words.

Literature Review

The Bayesian network was proposed by Judea Pearl in 1985. Bayesian network is also called a belief network or directed acyclic graphical model, which is a kind of probability graphic model. The nodes in the Bayesian network represent variables. These variables are observable or hidden. Also, they might be unknown parameters. The variables that have causality will be connected with the directed line. The port with arrows points to the "children", and another port points to the "parents". Moreover, these two variables will generate a conditional probability. For example, if the node E can affect H directly, then "E—>H", the conditional probability of "E—>H" is written like this "P(H|E)" [1]. In one word, the Bayes network is the directed graph contains variables and conditional relationship. It mainly describes the conditional dependency between variables, using circles represents variables and the arrow lines for conditional dependencies. There are three types in the Bayes network, which is head-to-head, tail-to-tail and head-to-tail. Bayes network can stimulate the real world by the mathematic method. One of the applications of the Bayes network is the disease diagnosis [2].

Task Performance

Part 1

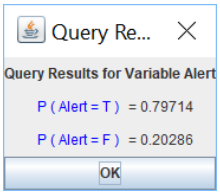
Based on the information provided, a total of five events are described. The phishing email detection system can determine the risk of the email being phishing email based on the content of the email. However, this detection system has a 3% chance of providing the wrong information. In this event, there should be two points corresponding to the phishing email detection system and mail type in the Bayesian network. The reason for using a dot to indicate the type of mail is that not all mail is phishing, and the detection system should not send out dangerous information and activate an alarm because of normal mail. For mail type points, it has two possibilities, one is normal mail, and the other is phishing mail. Moreover, the probability of occurrence in each case is the same 50%. For the mail detection system, it has a 3% probability of sending an error message, which means that there is a 3% chance that the monitoring system will not Issue the correct danger alert when it detects a phishing email, resulting in an

alert that will not be activated. Therefore, in the Bayesian network, the mail type and the phishing detection system serve as a precondition for sensing dangerous information. The dangerous information is only perceived when the type of mail is phishing and the phishing detection system is working correctly. After the dangerous information is obtained, the alarm will sound an alarm. The second event is the firewall, which protects the data service. However, the firewall has a 5% chance of shutting down when the staff is repairing the firewall. Also, there are other risks in repairing the firewall. That is, when the repair time expires, a network vulnerability will be formed, and the probability is 2%. Two event points are available in the firewall event, one is to repair the firewall, and the other is the repair timeout. These two events serve as a precondition for the firewall to issue dangerous information. When the firewall is repaired, and the firewall is down, the system should issue a hazard message. Alternatively, when the maintenance of the firewall exceeds the repair time, causing a network vulnerability, in which case the system will also issue a dangerous alarm to activate the alarm. That is to say, any condition of firewall inactivation and repair timeout will issue a dangerous message and activate the alarm. The third event described an analysis of the data that the risk of a network being attacked during holidays and political events during the year. The problem assumes that the area where the CNX network is located has 100 days of holidays and 360 days of political events. Therefore, the probability of the corresponding holiday node and political event node in the Bayesian network is 100/365 and 360/365, respectively. Moreover, these two nodes will become the parents of the node that represented the risk in holiday and politic events. When the alarm receives a dangerous message, it does not necessarily give an alarm. The alarm has a 20% probability that it will not alert even if it senses the risk of being attacked. So regardless of whether the dangerous information comes from a phishing email detection system or a firewall, the alarm will only activate the alarm with a probability of 80%. The CNX network has a logging system to investigate attacks on the system. The way this system distinguishes logs is to tell if the activity is abnormal. If the activity is abnormal, this activity is recorded as an attack event. However, the logging system has a lower fault tolerance rate and a 30% probability of getting the wrong value. Therefore, the logging system can correctly record an attack event, when the information obtained by the log system analysis is correct, and the network is attacked. Otherwise, despite the attack on the network, the logging system will not record this activity because it did not differentiate the attack as an attack. Therefore, the alarm and the differentiate correctly is the parent of the logging system.

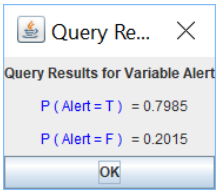
Based on the first network, the second network added two new events, downloading files and automatically updating system vulnerabilities. Most of the electricity is attacked by viruses because the files that carry the Trojans are downloaded. Usually, after the file is downloaded, the computer will automatically check if the file contains a Trojan. However, sometimes, some Trojans may not be detected and enter the computer causing hidden dangers. In this event, the probability of downloading the file is 10% for the Trojan, and the Trojan detection software can detect 80% of the Trojans. Therefore, the system will issue a virus danger message and activate the alarm, only if the downloaded file carries a Trojan and the Trojan detection software finds the Trojan in the file. Otherwise, the system will not issue virus danger information under other circumstances. Also, the second network added an automatic update event. The computer's program is continuously updated over time, and after a while, a patch to fix the vulnerability will appear. If the system vulnerability is dangerous, when the patch package appears, the system will automatically install the patch and update the system. However, patches for some vulnerabilities may require the user to select the installation manually. When a system vulnerability is not fixed, that is, when no patch is installed, the system will increase the risk of being attacked. The hypothesis

is the system will download 70% patches and update to protect the system from attacks. Moreover, if there are any bugs not fixed by patches, the system will have bug risk, and the alarm will be activated.

Question 1. What the probability of the alarm sends an alert in both networks? As the result shows in the picture, the probability of alarm is activated in the network two will be greater than the probability in network one.



-Network 1-



-Network 2-

Question 2. What is the probability of the risk not from the firewall when the alarm sends an alert in both networks? In the first network, there three kinds of risk, phishing email, firewall and special days. As for the second network, adding two new events, virus risk and bug risk. Table 1 shows all circumstances of risks and alarm. There are seven circumstances that alarm will send an alert. In these seven conditions, there are three times that the risk is not from the firewall. Therefore, in network 1, the answer to this question is 3/7, 17/31 in network 2.

Phishing Risk	Firewall Risk	Special-day Risk	Alert
T	T	T	T
F	T	T	T
T	F	T	T
T	T	F	T
F	F	T	T
T	F	F	T
F	T	F	T
F	F	F	F

- Table 1-

Question 3. Why is the probability of sending an alert in network one less than the probability in network 2? The reason is two more events can activate the alarm, that is, adding the circumstance that can make alarm send the alert. Thus, the probability in the network two is greater.

Part 2

Design

According to the guidance of the lecture, combining two Bayesian networks into one network. First, reading the Bayesian network in the XML file into the program. Creating an object for each point in the network and save it in an ArrayList. Each node object contains four attributes, name, for, given, and table after getting all the points of the two networks, starting to summarise the intersection, internal nodes and external nodes. Then sorting the dependencies of the different nodes, according to the rule to remove the dependency of the node. Finally, merging CPT and generating the new table.

Example and Testing

1. Read data from XML

To operate the nodes in the Bayesian network, the data in the Bayesian network needs to be read into the program and saved as objects. This step requires the help of the SAXReader tool in the dom4j package. SAXReader can get the root node in the XML file, and then iterate out the child node with the root node. Use conditional judgment to find the "VARIABLE" tag representing the node and create a node object using the "NAME" node value in "VARIABLE". Then, use the same method to find the "DEFINITION" node under the "NETWORK" node, and assign the values under this node to the node objects created in the previous step.

```
SAXReader reader = new SAXReader();//initialize the SAXReader tool
Document document = reader.read(new File(s));//read the data in the XML file
Element node = document.getRootElement();//get the root node of the network
Element Network = node.element("NETWORK");//get the deeper node by the root node
Iterator<Element> tt = Network.elementIterator();//Iterate the node in "NETWORK
//Finding the tag "VARIABLE" and creating the node object using the value in "NAME" tag
while (tt.hasNext()) {

	Element e = tt.next();
	if (e.getName().equals("VARIABLE")) {
		Iterator<Element> itt = e.elementIterator();
		while (itt.hasNext()) {
			Element ee = itt.next();
			if (ee.getName().equals("NAME")) {
				Node n = new Node(ee.getText());
				n1.add(n);
			}
		}
	}
}
```

The function will return the ArrayList used to stored the node objects. To test the function, printing all node's information in the ArrayList on the console. Following codes can test the validity of the function.

```
for(int i=0;i<n1.size();i++) {
	System.out.println("NAME " + n1.get(i).Name);
	System.out.println("FOR " + n1.get(i).For);
	for(int j=0;j<n1.get(i).Given.size();j++) {
		System.out.print("GIVEN " + n1.get(i).Given.get(j) + " ");
		System.out.println();
	}
	System.out.println("TABLE " + n1.get(i).Table);
	System.out.println();
}
```

2. Getting internal nodes and external nodes

Finding intersection and dividing the intersection in internal nodes and external nodes. The internal node is the node that its parents from one of the two networks is entirely contained in the intersection. If the node does not have any parent, that means the set of its parents is an empty set. In this case, this node is also an internal node because any set contains the empty set. On the other hand, the external node is the node that in the intersection but it is not an internal node. The following codes are the partial implementation of getting internal codes and external codes.

Getting the intersection of two networks initially.

```
ArrayList<Node> intersection = new ArrayList<Node>();
for (int i = 0; i < AllNode.size(); i++) {
	if (n1.contains(AllNode.get(i)) && n2.contains(AllNode.get(i))) {
		intersection.add(AllNode.get(i));
	}
}
```

Then finding the internal nodes from the first Bayes network 1.
Getting the intersection nodes in network 1 and recognising the number of its parents. If the node does not have parents which means the set of the parent is an empty set, then, adding the node into the internal list. Also, if all its parents are in the intersection, then adding the node as well. The parameter "test" is to count the number of the node's parent that is in the intersection. If the count equals the number of node's parent, that is, all its parents are in the intersection.

```
if (n1.get(j).Name.equals(name)) {
	int parent = n1.get(j).Given.size();
	//in the case of no parent
	if (parent == 0) {
		internal.add(n1.get(j));
	} else {
	//in the case of having parents and recognise if the set of the parent is in intersection
		for (int p = 0; p < parent; p++) {
			Node Parent = new Node(n1.get(j).Given.get(p));
			int test = 0;
			if (intersection.contains(Parent)) {
				test++;
			}
			if (test == parent) {
				internal.add(n1.get(j));
			}
		}
	}
}
```

In the same way, collecting the internal nodes from network 2. However, it should avoid adding repeated nodes into internal list. To test the validity of the function, the following code can print all internal nodes and the number of the internal node list.

```
for(int i=0;i<internal.size();i++) {
	System.out.println("internal " + internal.get(i).Name);
}
System.out.println("internal " + internal.size());
```

As for external nodes, removing the non-internal node in the intersection, and the rest of nodes are external nodes.

```
for (int i = 0; i < intersection.size(); i++) {
	if (!internal.contains(intersection.get(i))) {
		external.add(intersection.get(i));
	}
}
```

3. Modify Dependencies

The node in two networks has different dependencies. Thus, the same node might have many dependencies from the same or different parents. In this step, modifying the dependencies, which means removing unwanted dependencies based on rules. The delete rule is to keep the dependency in the network which the node's parent is not in the intersection. Otherwise, keep the dependency in the case that the node has more parents. The following codes are the implementation for getting all dependencies in all nodes, although it is not necessary. The dependencies will be described in the character array, and the character array will be stored in a list.

```
for (int i = 0; i < n1.size(); i++) {
	if (n1.get(i).Given.size() != 0) {
		int parent_num = n1.get(i).Given.size();
		String[] dependency = new String[parent_num + 1];
		for (int j = 0; j < parent_num; j++) {
			dependency[j] = n1.get(i).Given.get(j);
		}
		dependency[dependency.length - 1] = n1.get(i).Name;
		dependencies.add(dependency);
	}
}
```

By the similar function, getting the dependencies of both networks. The following codes show the method of getting the dependency on network 1. Using the same method to the network 2, and getting the internal nodes' dependencies.

```
for (int i = 0; i < n1.size(); i++) {
	if (n1.get(i).Given.size() != 0 && internal.contains(n1.get(i))) {
		int parent_num = n1.get(i).Given.size();
		String[] dependency = new String[parent_num + 1];
		for (int j = 0; j < parent_num; j++) {
			dependency[j] = n1.get(i).Given.get(j);
		}
		dependency[dependency.length - 1] = n1.get(i).Name;
		dependency_inter.add(dependency);
	}
}
```

At this time, testing the internal nodes' dependencies by the following code. Moreover, there are repeated dependencies and unwanted dependencies in the test result.

```
for(int i=0;i<dependency_inter.size();i++) {
	for(int j=0;j<dependency_inter.get(i).length;j++) {
		System.out.print(dependency_inter.get(i)[j]+" ");
	}
	System.out.println();
}
```

```
Test result:
Influenza Sore Throat
Influenza Smokes Bronchitis
Bronchitis Wheezing
Bronchitis Coughing
```

Smokes Sore Throat
Sore Throat Bronchitis
Bronchitis Wheezing
Bronchitis Asthma Coughing

After obtaining all dependencies of internal nodes, deleting unwanted dependencies based on the delete rule. The delete operation cannot execute directly by traversing the list of dependencies, because once deleting an element of the list, the order of other elements in the list will change. This question will cause that delete the wrong element. Therefore, storing the index of the element which is unwanted in an index list, after traversing the whole dependencies list. Then, picking up the element that the index is not in the index list of the dependencies list, as a result creating a new list that only contains the dependencies are wanted. Following is the part of the implement codes.
Deleting the repeated dependencies.

```
if (array.length == array_searched.length && s.equals(s_searched)
                        && array[0].equals(array_searched[0])) {
            sign.add(i);
}
```

The following codes aimed to compare parents. Initially, keeping the dependency that has more parents, and storing the index of unwanted dependency. Moreover, if the number of the parents are same, then keeping the dependency that the node's parents are not in the intersection nodes and storing the index of another dependency.

```
if (s.equals(s_searched)) {
            //sign the dependency that has less parent
      if (array.length > array_searched.length) {
            if (!sign.contains(j)) {
                  sign.add(j);
            }
            } else if (array_searched.length > array.length) {
                  sign.add(i);
            } else if (array.length == array_searched.length) {
            //sign the dependency that the parent is in the intersection
                  Node n1_test = new Node(array[0]);
                  Node n2_test = new Node(array_searched[0]);
                  if (intersection.contains(n1_test)) {
                        if (!sign.contains(i)) {
                              sign.add(i);
                        }
                  } else if (intersection.contains(n2_test)) {
                        if (!sign.contains(j)) {
                              sign.add(j);
                  }
            }
      }
}
```

Creating a new list to store the necessary dependencies, and update the internal dependencies list at last.

```
ArrayList<String[]> replace = new ArrayList<String[]>();
	for (int i = 0; i < dependency_inter.size(); i++) {
		if (!sign.contains(i)) {
			replace.add(dependency_inter.get(i));
		}
	}
dependency_inter = replace;
```

Now, re-print out the internal nodes' dependencies again, the result does not contain the unwanted dependencies any more.

```
for(int i=0;i<dependency_inter.size();i++) {
	for(int j=0;j<dependency_inter.get(i).length;j++) {
		System.out.print(dependency_inter.get(i)[j]+" ");
	}
	System.out.println();
}
```

Test result:
Influenza Sore Throat
Influenza Smokes Bronchitis
Bronchitis Wheezing
Bronchitis Asthma Coughing

Next, collecting the external nodes' dependencies. The method is similar to the method using in collecting internal nodes' dependencies. Following codes are the example that is getting the external nodes' dependencies in network 1.

```
for (int i = 0; i < n1.size(); i++) {
	if (n1.get(i).Given.size() != 0 && external.contains(n1.get(i))) {
		int parent_num = n1.get(i).Given.size();
		String[] dependency = new String[parent_num + 1];
		for (int j = 0; j < parent_num; j++) {
			dependency[j] = n1.get(i).Given.get(j);
		}
		dependency[dependency.length - 1] = n1.get(i).Name;
		dependency_exter.add(dependency);
	}
}
```

To generate the new table, getting the parent objects of the node from two networks.

```
for (int i = 0; i < n1.size(); i++) {
	for (int j = 0; j < external.size(); j++) {
		if (n1.get(i).equals(external.get(j))) {
			Node n = n1.get(i);
			Combine_external.add(n);
		}
	}
```

```
}
```

Using these codes to test the function:

```
for(int i=0;i<Combine_external.size();i++) {
	System.out.print("Name: " + Combine_external.get(i).Name);
	System.out.print("  For: " + Combine_external.get(i).For);
	for(int j=0;j<Combine_external.get(i).Given.size();j++) {
	System.out.print("  Given: " + Combine_external.get(i).Given.get(j));
	}
	System.out.print("  Table: " + Combine_external.get(i).Table);
	System.out.println();
}
```

The test result is like this:
Name: Fever For: Fever Given: Influenza Table: 0.9 0.1 0.05 0.95
Name: Fever For: Fever Given: Cold Table: 0.8 0.2 0.3 0.7
Therefore, the table of two parents has been got. The next step is to combine two parents and compute the new table.
Putting every parent's table value into a character array and store in the list.

```
ArrayList<String[]> table_value = new ArrayList<String[]>();
for (int i = 0; i < Combine_external.size(); i++) {
	String[] table = new String[Combine_external.get(i).Table.split("
").length];
	table = Combine_external.get(i).Table.split(" ");
	table_value.add(table);
}
```

Computing the new table value based on the formula p = p1+p2-p1*p2. Thus, to compute the probability of the fever in the condition that influenzas is true and the cold is true as well. The responding codes should be like this:

```
new_table[0] = Double.parseDouble(table_value.get(0)[0]) +
Double.parseDouble(table_value.get(1)[0])
                          - Double.parseDouble(table_value.get(0)[0]) *
Double.parseDouble(table_value.get(1)[0]);
```

The following numbers show the value of the new table.
0.9800000000000001, 0.28, 0.9299999999999999, 0.73, 0.81, 0.96,
0.33499999999999996, 0.985,
After normalising, the table will change like this:
0.7777777777777777, 0.2222222222222222, 0.5602409638554217,
0.43975903614457834, 0.4576271186440678, 0.5423728813559322,
0.2537878787878788, 0.7462121212121213,

Evaluation

The construction of the program is completed according to the guidance of the powerpoint guidance. Although the expected result of each step is obtained, it may be that the two Bayesian networks are not truly merged. The reason may be that the information obtained is not built into Bayeux by using Encog or other data structures.

Running

To run the program, put the command line “java -jar prob.jar example1.xml example2.xml” in the terminal. The important thing is the XML files (example1.xml, example2.xml) have to be stored in the same folder with the JAR program. Otherwise, the program cannot find the XML files, and the program will not run usually.

Bibliography

1. Friedman N, Geiger D, Goldszmidt M. Bayesian network classifiers. Machine learning. 1997 Nov 1;29(2-3):131-63.
2. Agosta JM, Gardos T. Bayes Network" Smart" Diagnostics. Intel Technology Journal. 2004 Nov 1;8(4).