Gabriel Brehm
CS361
June 2, 2019

Lab 4 Report

Part 1

First, I created a State class for my deterministic finite automaton (DFA). Each State object has a field for a string label, which is just used for console outputs, and a Boolean value to determine it the given state is an accept state or not.

```
class State
{
        private string label;

        private bool acceptState;
}
```

Then, I needed a DFA class, which consists of five fields. First, there is the list of states. Then there is the State that is specified to be the starting State. Third is the list of accepting States. Forth is the list of chars which are valid for this DFA, called the alphabet. Last is the delta table, which is a two-dimensional array of States.

```
class DFA
{
        private List<State> states;

        private State startState;

        private List<State> acceptingStates;

        private List<char> alphabet;

        private State[,] deltaTable;
}
```

Now we have the meat and potatoes of the DFA class, and that is the IsValidString method, which determines if a given string is valid for the current DFA object. The first thing the method does is checks if all the characters of the given string are in the alphabet. If there exists even one that is not in the alphabet, the method returns false. Then an indexer, i, is initialized to zero, and a temporary state is set to the starting State. We then enter a while loop that continues until the indexer is equal to the length of the input string. Nested in there is a for loop that goes from $j = 0$ to the number of characters in the alphabet. Nested in there is yet another for loop that goes from $k = 0$ to the number of States in the DFA. Then if the current state is equal to the kth State and if the ith character of the given string is equal to the jth character in the alphabet, then the temp State is set equal to the delta table of $k, j + 1$. Then once the looping has ceased, the temp State's acceptState Boolean value is returned.

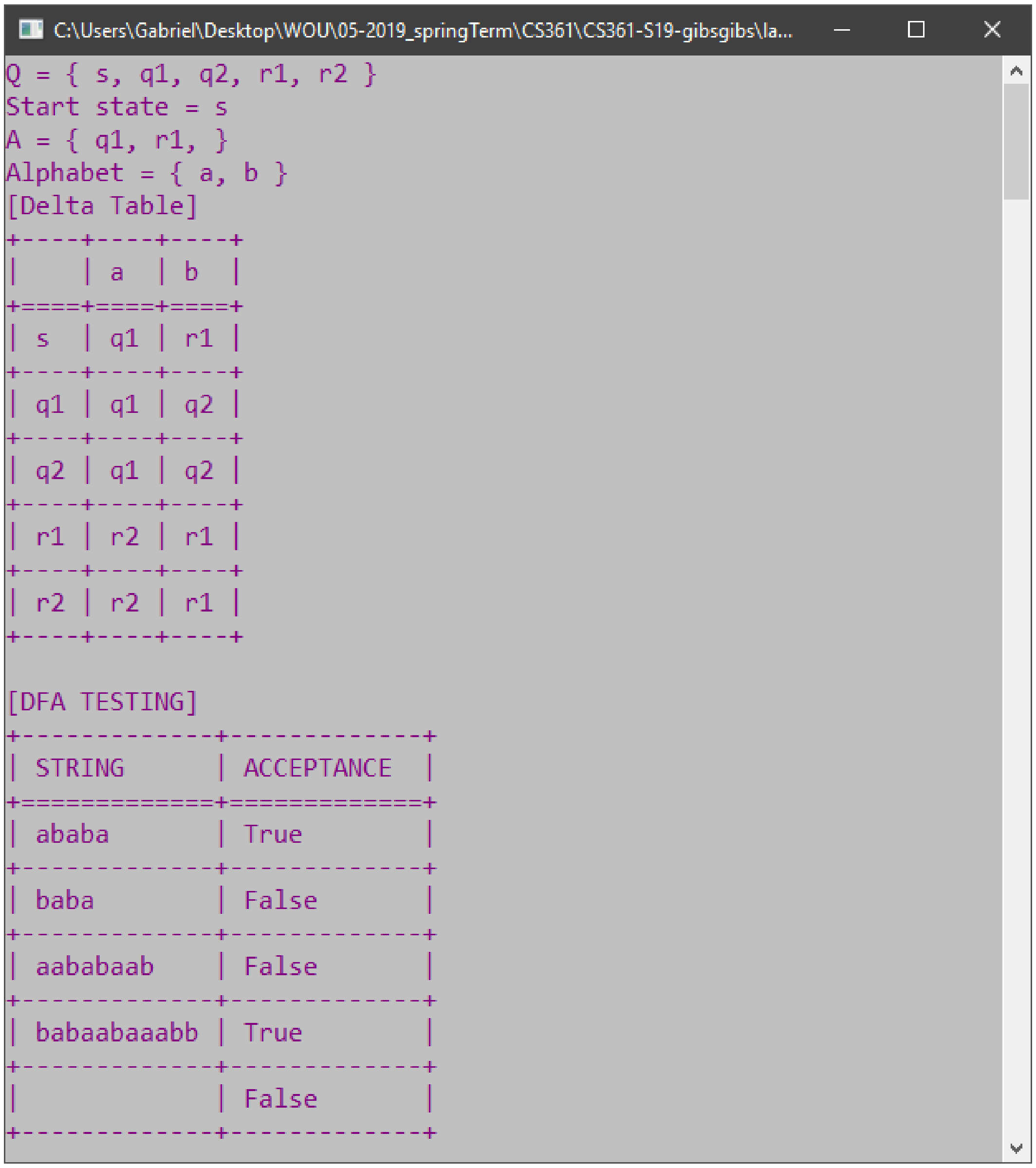
```
public bool IsValidString(string input)
{
        for(int j = 0; j < input.Length; j++)
        {
                if(!alphabet.Contains(input[j]))
                {
                        return false;
                }
        }
        int i = 0;

        State currentState = startState;

        while (i < input.Length)
        {
                for (int j = 0; j < alphabet.Count; j++)
                {
                        for (int k = 0; k < states.Count; k++)
                        {
                                if (currentState == states[k] && input[i] == alphabet[j])
                                {
                                        currentState = deltaTable[k, j + 1];
                                }
                        }
                }
                i++;
        }
        return currentState.GetAcceptState();
}
```

Part 2

Below is a screenshot showing the console output of the attributes of the given DFA, as well as the output of each of the five given strings.



C:\Users\Gabriel\Desktop\WOU\05-2019_springTerm\CS361\CS361-S19-gibsgibs\la...

```
Q = { s, q1, q2, r1, r2 }
Start state = s
A = { q1, r1, }
Alphabet = { a, b }
[Delta Table]
+----+----+----+
|    | a  | b  |
+====+====+====+
| s  | q1 | r1 |
+----+----+----+
| q1 | q1 | q2 |
+----+----+----+
| q2 | q1 | q2 |
+----+----+----+
| r1 | r2 | r1 |
+----+----+----+
| r2 | r2 | r1 |
+----+----+----+

[DFA TESTING]
+------------+------------+
| STRING     | ACCEPTANCE |
+============+============+
| ababa      | True       |
+------------+------------+
| baba       | False      |
+------------+------------+
| aababaab   | False      |
+------------+------------+
| babaabaaabb | True      |
+------------+------------+
|            | False      |
+------------+------------+
```

Part 3

In order to implement the Bellman-Ford algorithm, I needed to be able to implement a graph. Mathematically, a graph is defined as a nonempty set of vertices and a set of edges. So, I decided to create a Vertex class, with three fields: a string label, an int distance, and a vertex parent.

```
class Vertex
{
        private string label;

        private int distance;

        private Vertex parent
}
```

I decided to create an Edge class as well, and define my Edges as two vertices with a weight between them. So, the three fields are a Vertex startVertex, a Vertex endVertex and an int weight.

```
class Edge
{
        private Vertex startVertex;

        private Vertex endVertex;

        private int weight;
}
```

Now I needed to create a Graph class that uses the vertices and edges defined above. The fields for a Graph are a string label, a Vertex array vertices, and an Edge array edges.

```
class Graph
{
        private string label;

        private Vertex[] vertices;

        private Edge[] edges;
}
```

Now the first of two helper methods for Bellman-Ford is the initialize method. This method takes in the given Graph and the Vertex s, which is the starting Vertex, and sets each Vertex's distance to int.MaxValue, and their parent to null. Then it sets s's distance to zero.

```
private static void InitializeSingleSource(Graph G, Vertex s)
{
        foreach (Vertex v in G.GetVertices())
        {
                v.SetDistance(int.MaxValue);

                v.SetParent(null);
        }
        s.SetDistance(0);
}
```

The second helper method is the Relax method. This one takes in a single Edge e, and if e's starting Vertex's distance is not equal to int.MaxValue, then it checks if e's ending Vertex's distance is greater than the starting Vertex's distance plus the weight between the two vertices. If that is true, then the ending Vertex's distance is set to the right hand side of the inequality, and its parent is set to the starting Vertex.

```
private static void Relax(Edge e)
{
        if (e.GetStartVertex().GetDistance() != int.MaxValue)
```

```
		{
			if (e.GetEndVertex().GetDistance() > (e.GetStartVertex().GetDistance() +
			e.GetWeight()))
			{
				e.GetEndVertex().SetDistance(e.GetStartVertex().GetDistance() +
				e.GetWeight());

				e.GetEndVertex().SetParent(e.GetStartVertex());
			}
		}
	}
```

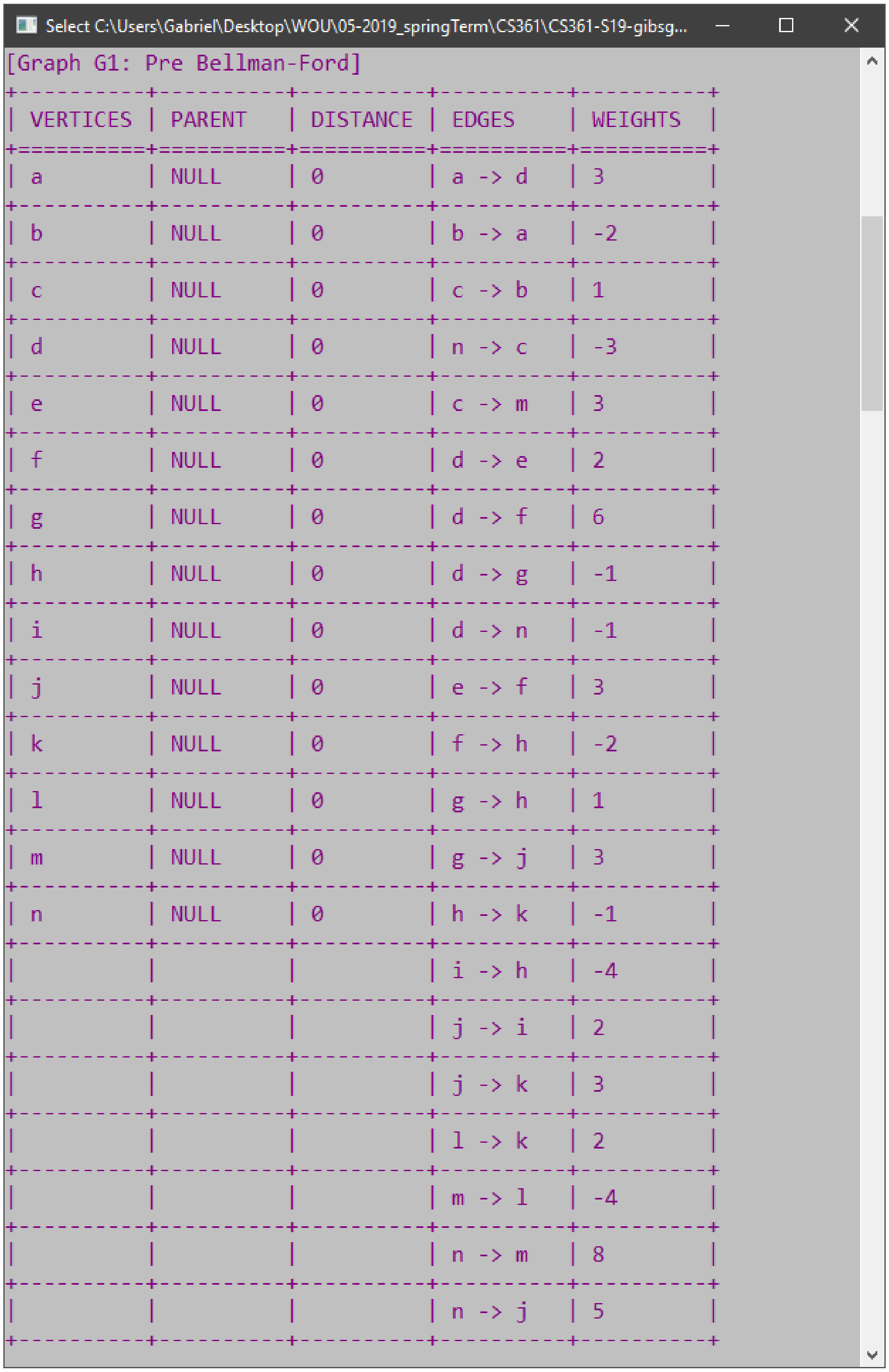
Finally, we get to the Bellman-Ford Algorithm. It takes in a Graph G, and a Vertex s, as the starting Vertex. First, it calls the initializing method defined above. Then, for each Vertex in G, we call Relax on each Edge. Once that has been completed, we do the same thing Relax does one more time. This time, however, if anything is altered, the method returns a string saying there is a negative weight cycle. If nothing is altered, then a string is returned saying there is no negative eight cycle.

```
	public static string BellmanFord(Graph G, Vertex s)
	{
		InitializeSingleSource(G, s);

		for (int i = 0; i < G.GetVertices().Length - 1; i++)
		{
			foreach (Edge e in G.GetEdges())
			{
				Relax(e);
			}
		}
		foreach (Edge e in G.GetEdges())
		{
			if (e.GetStartVertex().GetDistance() != int.MaxValue)
			{
				if (e.GetEndVertex().GetDistance() > (e.GetStartVertex().GetDistance() +
				e.GetWeight()))
				{
					return "Result of Bellman-Ford:\nNegative Weight Cycle Detected\n";
				}
			}
		}
		return "Result of Bellman-Ford:\nNo Negative Weight Cycle Detected\n";
	}
```

Part 4

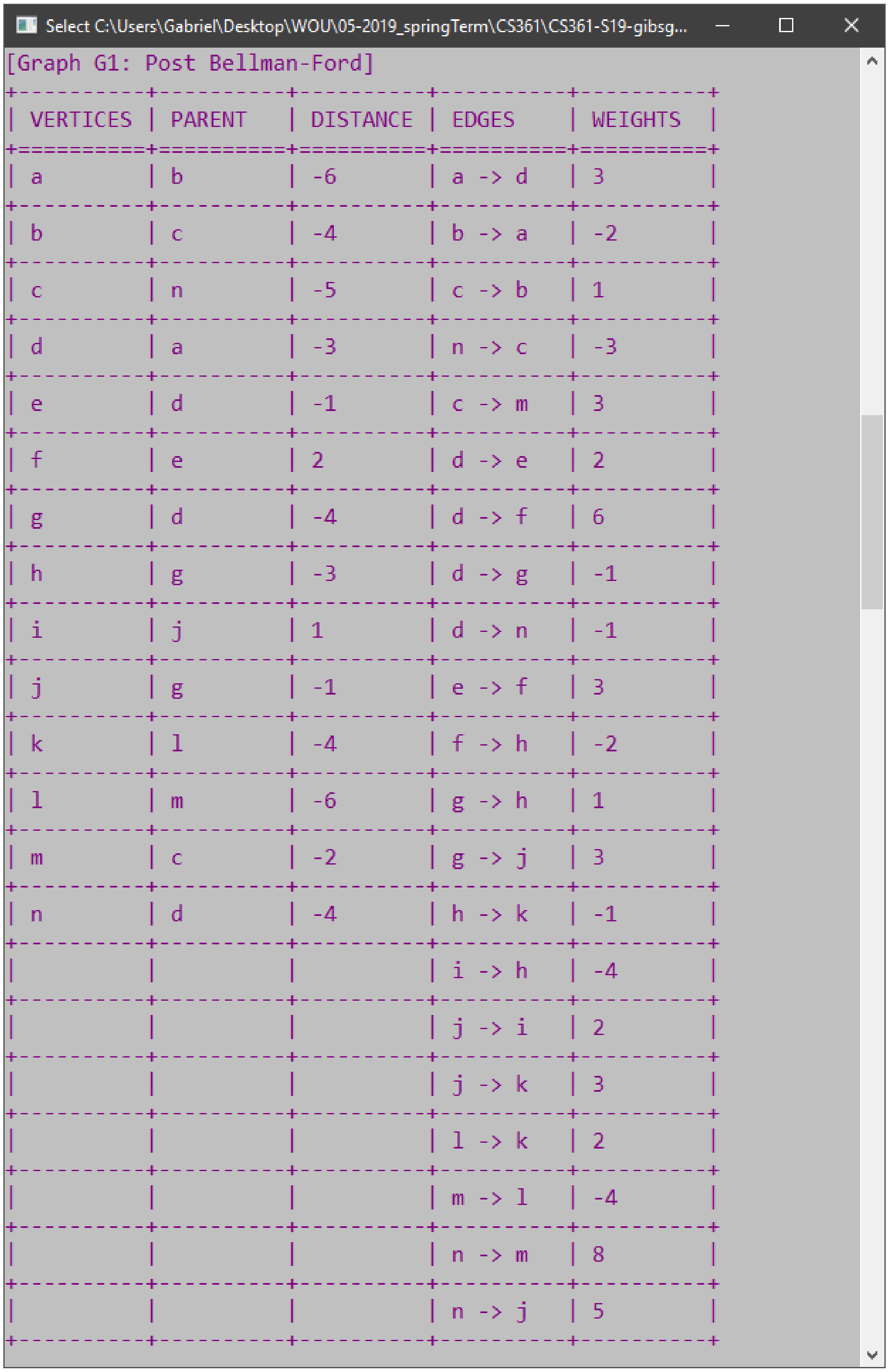
Here we can see all the information about the given Graph, before anything is done to it.



Select C:\Users\Gabriel\Desktop\WOU\05-2019_springTerm\CS361\CS361-S19-gibsg...

```
[Graph G1: Pre Bellman-Ford]
+----------+----------+----------+----------+----------+
| VERTICES | PARENT   | DISTANCE | EDGES    | WEIGHTS  |
+==========+==========+==========+==========+==========+
| a        | NULL     | 0        | a -> d   | 3        |
+----------+----------+----------+----------+----------+
| b        | NULL     | 0        | b -> a   | -2       |
+----------+----------+----------+----------+----------+
| c        | NULL     | 0        | c -> b   | 1        |
+----------+----------+----------+----------+----------+
| d        | NULL     | 0        | n -> c   | -3       |
+----------+----------+----------+----------+----------+
| e        | NULL     | 0        | c -> m   | 3        |
+----------+----------+----------+----------+----------+
| f        | NULL     | 0        | d -> e   | 2        |
+----------+----------+----------+----------+----------+
| g        | NULL     | 0        | d -> f   | 6        |
+----------+----------+----------+----------+----------+
| h        | NULL     | 0        | d -> g   | -1       |
+----------+----------+----------+----------+----------+
| i        | NULL     | 0        | d -> n   | -1       |
+----------+----------+----------+----------+----------+
| j        | NULL     | 0        | e -> f   | 3        |
+----------+----------+----------+----------+----------+
| k        | NULL     | 0        | f -> h   | -2       |
+----------+----------+----------+----------+----------+
| l        | NULL     | 0        | g -> h   | 1        |
+----------+----------+----------+----------+----------+
| m        | NULL     | 0        | g -> j   | 3        |
+----------+----------+----------+----------+----------+
| n        | NULL     | 0        | h -> k   | -1       |
+----------+----------+----------+----------+----------+
|          |          |          | i -> h   | -4       |
+----------+----------+----------+----------+----------+
|          |          |          | j -> i   | 2        |
+----------+----------+----------+----------+----------+
|          |          |          | j -> k   | 3        |
+----------+----------+----------+----------+----------+
|          |          |          | l -> k   | 2        |
+----------+----------+----------+----------+----------+
|          |          |          | m -> l   | -4       |
+----------+----------+----------+----------+----------+
|          |          |          | n -> m   | 8        |
+----------+----------+----------+----------+----------+
|          |          |          | n -> j   | 5        |
+----------+----------+----------+----------+----------+
```

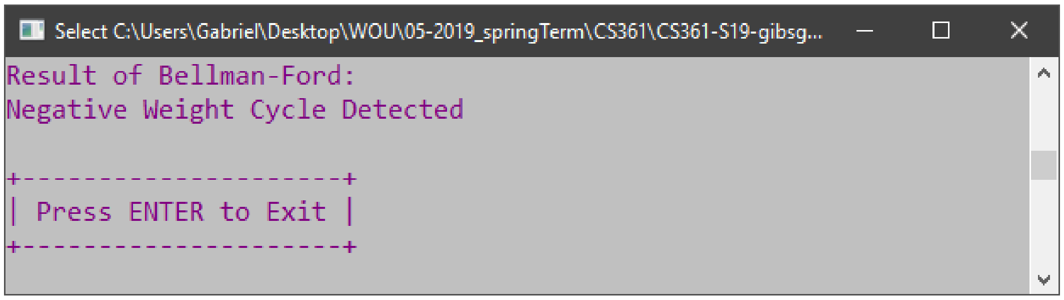
Below is the same given Graph, after the Bellman-Ford algorithm has been executed.



Select C:\Users\Gabriel\Desktop\WOU\05-2019_springTerm\CS361\CS361-S19-gibsg...

```
[Graph G1: Post Bellman-Ford]
+----------+----------+----------+----------+----------+
| VERTICES | PARENT   | DISTANCE | EDGES    | WEIGHTS  |
+==========+==========+==========+==========+==========+
| a        | b        | -6       | a -> d   | 3        |
+----------+----------+----------+----------+----------+
| b        | c        | -4       | b -> a   | -2       |
+----------+----------+----------+----------+----------+
| c        | n        | -5       | c -> b   | 1        |
+----------+----------+----------+----------+----------+
| d        | a        | -3       | n -> c   | -3       |
+----------+----------+----------+----------+----------+
| e        | d        | -1       | c -> m   | 3        |
+----------+----------+----------+----------+----------+
| f        | e        | 2        | d -> e   | 2        |
+----------+----------+----------+----------+----------+
| g        | d        | -4       | d -> f   | 6        |
+----------+----------+----------+----------+----------+
| h        | g        | -3       | d -> g   | -1       |
+----------+----------+----------+----------+----------+
| i        | j        | 1        | d -> n   | -1       |
+----------+----------+----------+----------+----------+
| j        | g        | -1       | e -> f   | 3        |
+----------+----------+----------+----------+----------+
| k        | l        | -4       | f -> h   | -2       |
+----------+----------+----------+----------+----------+
| l        | m        | -6       | g -> h   | 1        |
+----------+----------+----------+----------+----------+
| m        | c        | -2       | g -> j   | 3        |
+----------+----------+----------+----------+----------+
| n        | d        | -4       | h -> k   | -1       |
+----------+----------+----------+----------+----------+
|          |          |          | i -> h   | -4       |
+----------+----------+----------+----------+----------+
|          |          |          | j -> i   | 2        |
+----------+----------+----------+----------+----------+
|          |          |          | j -> k   | 3        |
+----------+----------+----------+----------+----------+
|          |          |          | l -> k   | 2        |
+----------+----------+----------+----------+----------+
|          |          |          | m -> l   | -4       |
+----------+----------+----------+----------+----------+
|          |          |          | n -> m   | 8        |
+----------+----------+----------+----------+----------+
|          |          |          | n -> j   | 5        |
+----------+----------+----------+----------+----------+
```

This is the string returned from the Bellman-Ford algorithm.



Select C:\Users\Gabriel\Desktop\WOU\05-2019_springTerm\CS361\CS361-S19-gibsg...

```
Result of Bellman-Ford:
Negative Weight Cycle Detected

+---------------------+
| Press ENTER to Exit |
+---------------------+
```

References

I worked with Jacob Malmstadt on this project.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). Cambridge, MA: The MIT Press.