Tyler Bartlett

CS361 Lab 4

This Lab started out easy for me when I did the DFA. I made a “state” object, then made many instances of the states, and populated the fields with the given instructions from the lab handout. Below is a screen shot of the state object and an example of how each state was setup, but with correct values for that state.

```
// A struct to hold state information
struct State
{
    bool startState = false;    // Denotes if a state is the start state
    bool acceptState = false;   // Denotes if a state is an accept state
    struct State* stateOnInputA = NULL;     // The state to move to if input = 'a'
    struct State* stateOnInputB = NULL;     // The state to move to if input = 'b'
};
```

```
struct State start, q1, q2, r1, r2;
start.startState = true;    // Start state is the starting state. much wow
start.acceptState = true;   // Start state accept empty string
start.stateOnInputA = &q1;  // The state to go to on input 'a' from start
start.stateOnInputB = &r1;  // The state to go to on input 'b' from start
```

I then checked the input from the strings and moved a pointer through the DFA from the start state to the correct state on its given input. With the DFA, I assumed input was checked prior to strings being sent to the DFA for analysis.

```
State* statePtr = &start;  // A pointer to keep track of what state we are at in the DFA

for (unsigned int i = 0; i < input.length(); i++)
{
    if (input[i] == 'a')
    {
        statePtr = statePtr->stateOnInputA;
    }
    else if (input[i] == 'b')
    {
        statePtr = statePtr->stateOnInputB;
    }
    else
    {
        //std::cout << "ERROR: invalid input detected!" << std::endl;
        //return -1;
    }
}

if (statePtr->acceptState == true)
{
    std::cout << "Input string: '" << input << "' was accepted." << std::endl;
    return 0;
}
else
{
    std::cout << "Input string: '" << input << "' was NOT accepted." << std::endl;
    return -1;
}
```

Mike Dietrich had provided me with his strings build and function calls to test them on the DFA which I used as a time save to not have to type it all out. Below is an example of that and the outputs of each string to test from the lab handout.

```
// The below block for testing was given to me by Mike and slightly modified to fit my use.
std::cout << "     Testing my DFA with the strings provided by professor for testing\n";
std::cout << "*********************************************************************************\n\n";
std::string testString1forDFA = { 'a','b','a','b','a' };              // Should accept.
std::string testString2forDFA = { 'b','a','b','a' };                  // Shouldn't accept.
std::string testString3forDFA = { 'a','a','b','a','b','a','a','b' };  // Shouldn't accept.
std::string testString4forDFA = { 'b','a','b','a','a','b','a','a','a','b','b' }; // should accept.
std::string testString5forDFA = { ' ' };                              // Should accept empty string.
std::cout << "First string being tested:  'ababa'        --- (Should Accept)\n";
runDFA(testString1forDFA);                                    // Testing first given string.
std::cout << "Second string being tested: 'baba'         --- (Should NOT Accept) \n";
runDFA(testString2forDFA);                                    // Testing second given string.
```

```
     Testing my DFA with the strings provided by professor for testing
*********************************************************************************

First string being tested:  'ababa'         --- (Should Accept)
Input string: 'ababa' was accepted.
Second string being tested: 'baba'          --- (Should NOT Accept)
Input string: 'baba' was NOT accepted.
Third string being tested:  'aababaab'      --- (Should NOT Accept)
Input string: 'aababaab' was NOT accepted.
Fourth string being tested: 'babaabaaabb'   --- (Should Accept)
Input string: 'babaabaaabb' was accepted.
Fifth string being tested:  ''              --- (Should Accept)
Input string: ' ' was accepted.

*********************************************************************************
```

Next up was the task of the Bellman Ford Algorithm, which quite frankly proved more troublesome than anticipated. I decided to make an edge object to represent each edge, a vertex object to do the same, and a graph to hold the information as seen in the below screen shots. I had decided that giving each vertex an array of edges would be good to track this information and how I had it done in my head, seemed to make the most sense. Jacob McLeod and I worked on writing the Bellman Ford algorithm together but hit a brick wall. Vertex K proved to be the bane of our existence because its array of edges is zero which completely broke the logic we had written which was based off the edge array at each vertex. Due to time constraints I could not improve much further upon our algorithm, but I got it to work correctly for each vertex other than Vertex K. with our relax function, it was not originally working with temp pointers so I brought it back into the same loop and broke it down by the objects pointers next vertices and what not. I have a comment in the code about an automagic fix to make declarations work because they for some reason didn't work in the "relax portion" of the for loop. The automagic fix finds the needed distance and predecessor information buried in the objects edge vector data. I don't know why it needed this crazy band aid but it worked, kind of. Except for vertex K.

```
struct Vertex
{
    std::vector<struct Edge> edgeVect;  // A vector to hold all edges from this vertex to another vertex
    int distance = INT_MAX;             // Default distance for a node is infinity
    struct Vertex* predecessor = NULL;  // Predecessor vertex
    char vertName;
};


struct Edge
{
    Vertex* source = NULL;       // The vertex this edge originates from
    Vertex* destination = NULL;  // The vertex this edge is going to from the source
    int weight;
};

struct Graph
{
    int numVertices, numEdges;      // Total number of vertices and edges in the graph
    std::vector<Vertex> vertices;   // A list of vertices in the graph
    Vertex* root = NULL;            // The source vertex in which calculations will be done from
};
```

The below four screen shots are examples of building the objects.

```
// Create all needed vertices based off graph given in lab instructions. Verticies are [a..n]
Vertex vertA, vertB, vertC, vertD, vertE, vertF, vertG, vertH, vertI, vertJ, vertK, vertL, vertM, vertN;
vertA.vertName = 'A'; vertB.vertName = 'B'; vertC.vertName = 'C';
```

```
// Create all edges based off graph given in lab instructions
Edge edgeAtoD;
edgeAtoD.source = &vertA;
edgeAtoD.destination = &vertD;
edgeAtoD.weight = 3;
```

```
//populate vertex edge vector for each V based on instructions given in lab instructions.
vertA.edgeVect.push_back(edgeAtoD);
vertB.edgeVect.push_back(edgeBtoA);
vertC.edgeVect.push_back(edgeCtoB);
```

```
// Add vectors to graph
graph->vertices.push_back(vertA);
graph->vertices.push_back(vertB);
```

The next below screen shot is a portion of the bellman ford which mostly worked. I put a few different band aids in the loops to get around the issue caused by the automagic not magically working, then working with more automagic and to get around vertex K.

```
for (int i = 0; i <= graph->numVertices - 1; i++)
{
    // Fixes shit for verticies with no out-bound edges. vertK in this instance
    // makes some temp verticies, makes an edge between them and assigns that edge to vertK
    if (graph->vertices[i].edgeVect.size() == 0)
    {
        Vertex tempVertU, tempVertV;
        Edge edgeUtoV;
        edgeUtoV.destination = &tempVertV;
        edgeUtoV.source = &tempVertU;
        graph->vertices[i].edgeVect.push_back(edgeUtoV);

    }

    // FUCKING AUTOMAGIC FOR THE WIN (ノಠ益ಠ)ノ彡┻━┻. Don't ask how, I don't know why.
        graph->vertices[i].distance = graph->vertices[i].edgeVect[0].source->distance;
        graph->vertices[i].predecessor = graph->vertices[i].edgeVect[0].source->predecessor;

    for (unsigned int j = 0; j <= graph->vertices[i].edgeVect.size() - 1; j++)
    {
        // For some reason, vertx weight and predecessor are not being set. see the automagic fix lines above. IDFK ¯\_(ツ)_/¯
        if (graph->vertices[i].distance != INT_MAX && graph->vertices[i].edgeVect[j].destination->distance > graph->vertices[i].distance + graph->vertices[i].edgeVect[j].weight)
        {
            graph->vertices[i].edgeVect[j].destination->distance = graph->vertices[i].distance + graph->vertices[i].edgeVect[j].weight;
            graph->vertices[i].edgeVect[j].destination->predecessor = graph->vertices[i].edgeVect[j].source;
        }
```

And the last screen shot here is the output of the Bellman Ford algorithm, except with vertex K not displaying correctly. The correct values for K should be distance 2 and Predecessor vertH based of my hand trace of the algorithm.

```
Vertex:   Distance from Source:  Predessesor:
vertA,     distance 0,            This is the source
vertB,     distance Infinite,     Isolated Vertex
vertC,     distance Infinite,     Isolated Vertex
vertD,     distance 3,            vertA
vertE,     distance 5,            vertD
vertF,     distance 8,            vertE
vertG,     distance 2,            vertD
vertH,     distance 3,            vertG
vertI,     distance 7,            vertJ
vertJ,     distance 5,            vertG
vertK,     distance Infinite,     Isolated Vertex
vertL,     distance Infinite,     Isolated Vertex
vertM,     distance 10,            vertN
vertN,     distance 2,            vertD
```