Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica #4

Autómata no Determinístico

Axel Treviño Palacios
2CM5
09 de enero de 2020

1. Objetivo

Programar el autómata finito determinístico que reconozca las palabras:

- ebay
- web
- website
- webpage
- webmaster
- else
- we
- webay

2. Proceso de conversión

El mismo que la práctica pasada, exceptuando que creé un algoritmo para generar automáticamente el diagrama de nodos.

Name	isFinal	e	l	s	b	a	y	w	m	t	r	i	p	g
0	0	0, 1						0, 8						
0, 1	0	0, 1	0, 2		0, 5			0, 8						
0, 2	0	0, 1		0, 3				0, 8						
0, 3	0	0, 1, 4						0, 8						
0, 1, 4	1	0, 1	0, 2		0, 5			0, 8						
0, 5	0	0, 1				0, 6		0, 8						
0, 6	0	0, 1					0, 7	0, 8						
0, 7	1	0, 1						0, 8						
0, 8	0	0, 1, 9						0, 8						
0, 1, 9	1	0, 1	0, 2		0, 5, 10			0, 8						
0, 5, 10	1	0, 1		0, 17		0, 6, 25		0, 8	0, 11				0, 21	
0, 11	0	0, 1				0, 12		0, 8						
0, 12	0	0, 1		0, 13				0, 8						
0, 13	0	0, 1						0, 8		0, 14				
0, 14	0	0, 1, 15						0, 8						
0, 1, 15	0	0, 1	0, 2		0, 5			0, 8			0, 16			
0, 16	1	0, 1						0, 8						
0, 17	0	0, 1						0, 8				0, 18		
0, 18	0	0, 1						0, 8		0, 19				
0, 19	0	0, 1, 20						0, 8						
0, 1, 20	1	0, 1	0, 2		0, 5			0, 8						
0, 21	0	0, 1				0, 22		0, 8						
0, 22	0	0, 1						0, 8						0, 23
0, 23	0	0, 1, 24						0, 8						
0, 1, 24	1	0, 1	0, 2		0, 5			0, 8						
0, 6, 25	0	0, 1					0, 7, 26	0, 8						
0, 7, 26	1	0, 1						0, 8						

Figura 1: Tabla de Conversión

3. Código de autómata

Hay dos archivos, el archivo del servidor y el archivo de cliente.

```
{% load static %}
const grafo = require("./modules/grafo");
const readline = require("readline");
const express = require("express");
const path = require("path");
const fs = require("fs");
const app = express();

const http = require("http").createServer(app);
const webPort = 8080;

const pathDiccionario = path.join(
  __dirname,
  "..",
  "archivos",
  "diccionario.txt"
);
const pathPrettyTree = path.join(__dirname, "..", "archivos", "tree.json");
const pathCounter = path.join(__dirname, "..", "archivos", "contador.txt");
const pathProceso = path.join(__dirname, "..", "archivos", "proceso.txt");
const pathChart = path.join(__dirname, "..", "archivos", "chart.json");
const pathInput = path.join(__dirname, "..", "archivos", "input.txt");

let treantTree = {};
let counter = {};
let q0;
let io;

// EJS INIT
// Set the view engine to ejs
app.set("view engine", "ejs");
// Set ejs files path
app.set("views", __dirname + "/../dist/pages");

// REQUESTS
app.get("/", (req, res) => {
  res.render("index");
});
app.get("/get/tree/", (req, res) => {
  res.end(JSON.stringify(treantTree));
});
app.get("/generate/tree", (req, res) => {
  var response = {
    valid: false,
    message: "Archvio no encontrado!",
  };

  // Check if it exists
  try {
    if (fs.existsSync(pathDiccionario)) {
      response.valid = true;
      response.message = "Archivo cargado!";
    }
  } catch (err) {}
```

```
55
56    res.end(JSON.stringify(response));
57
58    generateTree();
59    generateTreantTree();
60  });
61  app.get("/start/", (req, res) => {
62    res.end(
63      JSON.stringify({
64        start: true,
65      })
66    );
67    processFile();
68  });
69
70  // NODE CREATOR
71  // Recursively adds a node
72  function addNode(node, word, value) {
73    let nodeName = value.substring(0, value.length - (word.length - 1));
74
75    if (word.length == 1) {
76      node.children[word] = new grafo.QNode(value, word, true);
77    } else {
78      // If child does not exist, create it.
79      if (typeof node.children[word[0]] == "undefined") {
80        node.children[word[0]] = new grafo.QNode(
81          nodeName,
82          word.substring(0, 1)
83        );
84      }
85
86      // Add children node
87      node.children[word[0]] = addNode(
88        node.children[word[0]],
89        word.substring(1),
90        value
91      );
92    }
93    return node;
94  }
95
96  // MAIN AUTOMATA
97  async function processFile() {
98    const readable = readline.createInterface({
99      input: fs.createReadStream(pathInput, { encoding: "utf8" }),
100   });
101   const writeStream = fs.createWriteStream(pathProceso);
102   let currentNodes = [q0];
103   let nextNodes;
104   let progress;
105   let message;
106   let limi;
107
108   console.log(`Reading file ${pathInput}...`);
```

```
109
110   for await (let line of readable) {
111     // Add space to end of the line
112     line += " ";
113     limi = line.length;
114
115     for (let i = 0; i < limi; ++i) {
116       // Empty needed arrays
117       nextNodes = [];
118       message = [];
119
120       // Reset progress
121       progress = `(${line[i]})=>\n`;
122
123       // Process all steps at the 'same time'
124       currentNodes.forEach((node) => {
125         // Add q0 to everything
126         nextNodes.push(q0);
127
128         // Add the rest of nodes
129         nextNodes.push(
130           node.process(line[i], q0, (node, char, next) => {
131             // Append progress to variable
132             progress += `\t${node.name} -> ${next}\n`;
133
134             // If it's an end node, add to counter
135             if (node.end) {
136               // If counter is nonexistant, create it
137               if (typeof counter[node.name] == "undefined") {
138                 counter[node.name] = 0;
139               }
140
141               // Add to counter
142               ++counter[node.name];
143
144               // Register counter
145               progress += `\t\t${node.name} +1\n`;
146             }
147
148             // Add to GUI list
149             message.push({
150               currentChar: char,
151               nodeIsEnd: node.end,
152               nodeStart: node.name,
153               nodeEnd: next,
154             });
155           })
156         );
157       });
158
159       // Write current step to web GUI
160       io.sockets.emit("update", JSON.stringify(message));
161
162       // Update counter on file
```

```
      writeCounter();

      // Write progress to file
      writeStream.write(progress);

      // Set the next nodes to be processed while also removing duplicate nodes
      currentNodes = [...new Set(nextNodes)];
    }
  }

  console.log("Done reading file!");
  writeStream.close();
}
function removeDuplicateNodes(nodeArray) {
  nodeArray.forEach((nodeArray) => {});
}
function writeCounter() {
  fs.writeFile(pathCounter, JSON.stringify(counter, null, 4), function () {});
}

// TREE FUNCTIONALITY
async function generateTree() {
  const readable = readline.createInterface({
    input: fs.createReadStream(pathDiccionario, { encoding: "utf8" }),
  });
  let fileTree;

  console.log("Generating tree from " + pathDiccionario);

  // Generate origin node
  q0 = new grafo.QNode("q0", "\0");

  // Create a node line for each word
  for await (let line of readable) {
    q0 = addNode(q0, line, line);
  }

  fileTree = fs.createWriteStream(pathPrettyTree);
  fileTree.write(JSON.stringify(q0, null, 4));
  fileTree.close();
}
function generateTreantTree() {
  treantTree.chart = JSON.parse(fs.readFileSync(pathChart));

  treantTree.nodeStructure = getGraphNode(q0);
}

function getGraphNode(node) {
  let result = {};
  let children = [];
  result.text = {
    name: node.name,
  };
  if (node.end) {
```

```
217       result.HTMLclass = "node-end";
218     }
219
220     for (let [key, val] of Object.entries(node.children)) {
221       children.push(getGraphNode(val));
222     }
223
224     result.children = children;
225     return result;
226   }
227
228   // INITIALIZERS
229   function initSocket() {
230     io = require("socket.io")(http);
231
232     // WEB SOCKET
233     io.on("connection", (socket) => {
234       var data = {
235         event: "handshake",
236         data: "Hola! C:",
237       };
238
239       console.log("Conexi n a socket!");
240       socket.emit("handshake", JSON.stringify(data));
241     });
242
243     console.log(io.path());
244
245     console.log(`Socket ready on ${http}`);
246   }
247   async function initTrees() {
248     // READ TREE SOURCE FILE
249     await generateTree();
250     generateTreantTree();
251     processFile();
252   }
253
254   // SERVER SET-UP
255   app.use(express.static(__dirname + "/../dist/public/"));
256
257   // INITIALIZE THINGS
258   initSocket();
259   initTrees();
260
261   // SERVER LISTEN INIT
262   http.listen(webPort, () => {
263     console.log("Listening on port: " + webPort);
264   });
```

```
{% load static %}
<!DOCTYPE html>
<html>

<head>
  <%- include('../partials/head'); %>

    <script type="module" defer>
      import { webSocket } from "/js/WebSocket.js";

      let updateQueue = new Queue();
      let waitTime = 100;
      let counter = {};

      webSocket.events = [
        {
          name: 'connect',
          handler: function () {
            customAlert("Conectado", 'success');
          }
        }, {
          name: 'disconnect',
          handler: function (reason) {
            customAlert("Desconectado!", 'error');
          }
        }, {
          name: 'handshake',
          handler: function (data) { }
        }, {
          name: 'update',
          handler: function (data) {
            // TODO recieve many node progresses
            updateQueue.enqueue(JSON.parse(data));
          }
        }
      ];


      // Graphic functions
      function updateCounter() {
        let head, body;

        $("#result-head").html("");
        $("#result-body").html("");

        for (let [key, val] of Object.entries(counter)) {
          head = document.createElement("th");
          $(head).prop("scope", "col");
          $(head).addClass("text-center");
          $(head).text(key);

          body = document.createElement("td");
          $(body).addClass("text-center");
          $(body).text(val);
```

```
55
56            $("#result-head").append(head);
57            $("#result-body").append(body);
58          }
59        }
60        function customAlert(text, mode = '', callback = () => {
61          $("#div-alert").fadeOut('fast');
62        }) {
63          $("#div-alert").fadeOut('fast', function () {
64
65            $("#div-alert").removeClass("alert-warning");
66            $("#div-alert").removeClass("alert-success");
67            $("#div-alert").removeClass("alert-danger");
68
69            switch (mode) {
70              case "success":
71                $("#div-alert").addClass("alert-success");
72                break;
73              case "error":
74                $("#div-alert").addClass("alert-danger");
75                break;
76              default:
77                $("#div-alert").addClass("alert-info");
78                break;
79            }
80
81            $("#div-alert").text(text);
82            $("#div-alert").fadeIn('slow', callback);
83          });
84        }
85        function drawRipple(x, y) {
86          let node = $(".ripple").clone();
87
88          node.addClass("animate");
89          node.css({
90            left: x,
91            top: y
92          });
93
94          $(".ripple").replaceWith(node);
95        };
96
97
98        // Node functions
99        function moveToNode(marker, node, time) {
100         return marker.animate({
101           width: node.outerWidth(),
102           height: node.outerHeight(),
103           left: node.offset().left,
104           top: node.offset().top,
105           opacity: '0.7'
106         }, time).promise();
107       }
108       async function animateNode(start, end, isEnd, time = waitTime, callback = () =>
```

```
109           let origin, dest;
110           let progress;
111           let marker = $("#progress-marker").clone().removeAttr("id").addClass(["step",
112
113           // Get nodes
114           origin = findNode(start);
115           dest = findNode(end);
116
117           // Add appropiate styling
118           marker.addClass(isEnd ? "bg-success" : "bg-danger");
119
120           // Move progress marker between nodes
121           await moveToNode(marker, origin, 0);
122           await moveToNode(marker, dest, time);
123           callback();
124         }
125         function findNode(text) {
126           return $("div").filter(function () {
127             return $(this).text() === text;
128           });
129         }
130
131
132         // Document ready
133         $(document).ready(function () {
134           $.get("/get/tree/", (data, status) => {
135             new Treant(JSON.parse(data), () => {
136               $(".node-end").addClass("bg-secondary");
137             });
138           });
139           webSocket.connect();
140
141           $("#form-control").submit(function (ev) {
142             ev.preventDefault();
143
144             counter = {};
145             updateCounter();
146
147             $.get("start/", function () { });
148           });
149
150           // Node animation dequeuing
151           setInterval(function () {
152             if (updateQueue.getLength() == 0) {
153               return;
154             }
155
156
157             // Remove all previous markers
158             $(".step").remove();
159
160             // Get next animation steps
161             let steps = updateQueue.dequeue();
162             let currentChar;
```

```
163             let endNode;
164
165             // Process steps simultaneously
166             steps.forEach(step => {
167               animateNode(step.nodeStart, step.nodeEnd, step.nodeIsEnd);
168
169               // Count words
170               if (step.nodeIsEnd) {
171
172                 // If the counter is null, create counter
173                 if (counter[step.nodeStart] == null) {
174                   counter[step.nodeStart] = 0;
175                 }
176                 ++counter[step.nodeStart];
177                 updateCounter();
178
179                 // Add ripple effect to node
180                 endNode = findNode(step.nodeStart);
181                 drawRipple(endNode.offset().left, endNode.offset().top);
182               }
183
184               currentChar = step.currentChar;
185             });
186
187             // Add character to progress 'bar'
188             $("#span-progress").text($("#span-progress").text().substring(1) + currentCh
189           }, waitTime);
190         });
191       </script>
192   </head>
193
194   <body class="d-block">
195     <div id="div-alert-container" class="d-flex justify-content-center fixed-top">
196       <div id="div-alert" class="alert text-justify" style="display: none;"></div>
197     </div>
198     <div class="container">
199       <div class="row">
200         <div class="d-flex w-100 flex-column">
201           <%- include('../partials/header'); %>
202
203             <main role="main" class="inner cover text-center w-100">
204               <h1 class="cover-heading">Grafo de Aut mata Webay</h1>
205               <p>
206                 Grafo representando a un aut mata que detecta varias palabras.
207               </p>
208               <form id="form-control">
209                 <button type="submit" class="btn-primary p-2 rounded">Iniciar</button>
210               </form>
211               <h3>Progreso:<br><span id="span-progress">
    </span></h3>
212               <br>
213             </main>
214         </div>
215       </div>
```

```
  </div>
  <div class="d-block">
    <div class="d-flex justify-content-center">
      <div class="col-auto">
        <table class="table table-responsive">
          <thead>
            <tr id="result-head"></tr>
          </thead>
          <tbody>
            <tr id="result-body"></tr>
          </tbody>
        </table>
      </div>
    </div>
    <div id="div-grafo-container" class="d-flex justify-content-center">
      <div id="div-grafo"></div>
    </div>
  </div>

  <!-- Ripple -->
  <div id="ripple-container" class="position-absolute w-100 h-100 overflow-hidden fixe
    <div class="ripple position-absolute overflow-hidden"></div>
  </div>

  <!-- Progress marker -->
  <div id="progress-marker" class="position-absolute p-3"></div>
</body>

</html>
```

4. Resultados

Autómata ejecutándose:

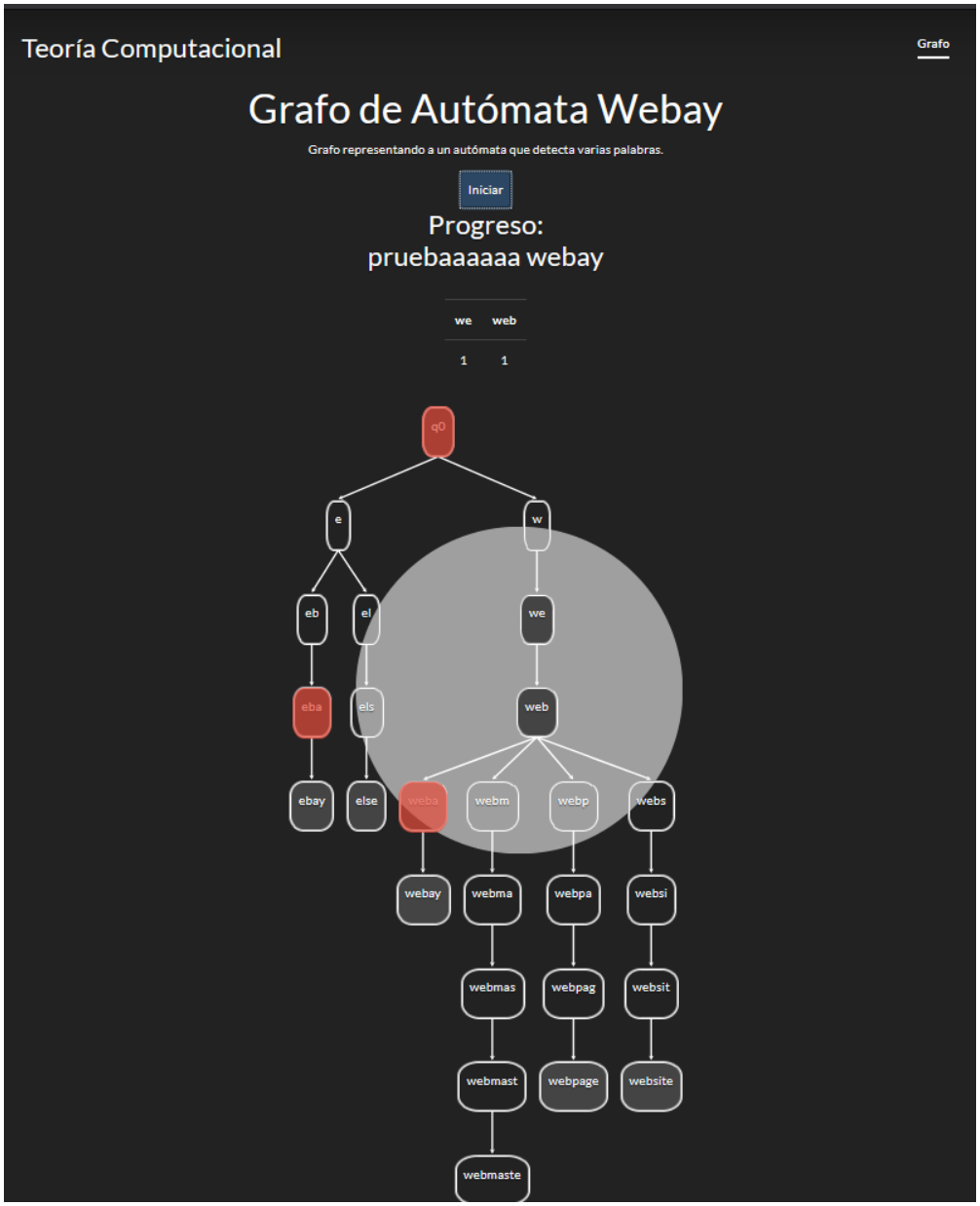

Figura 2: Tabla de Conversión

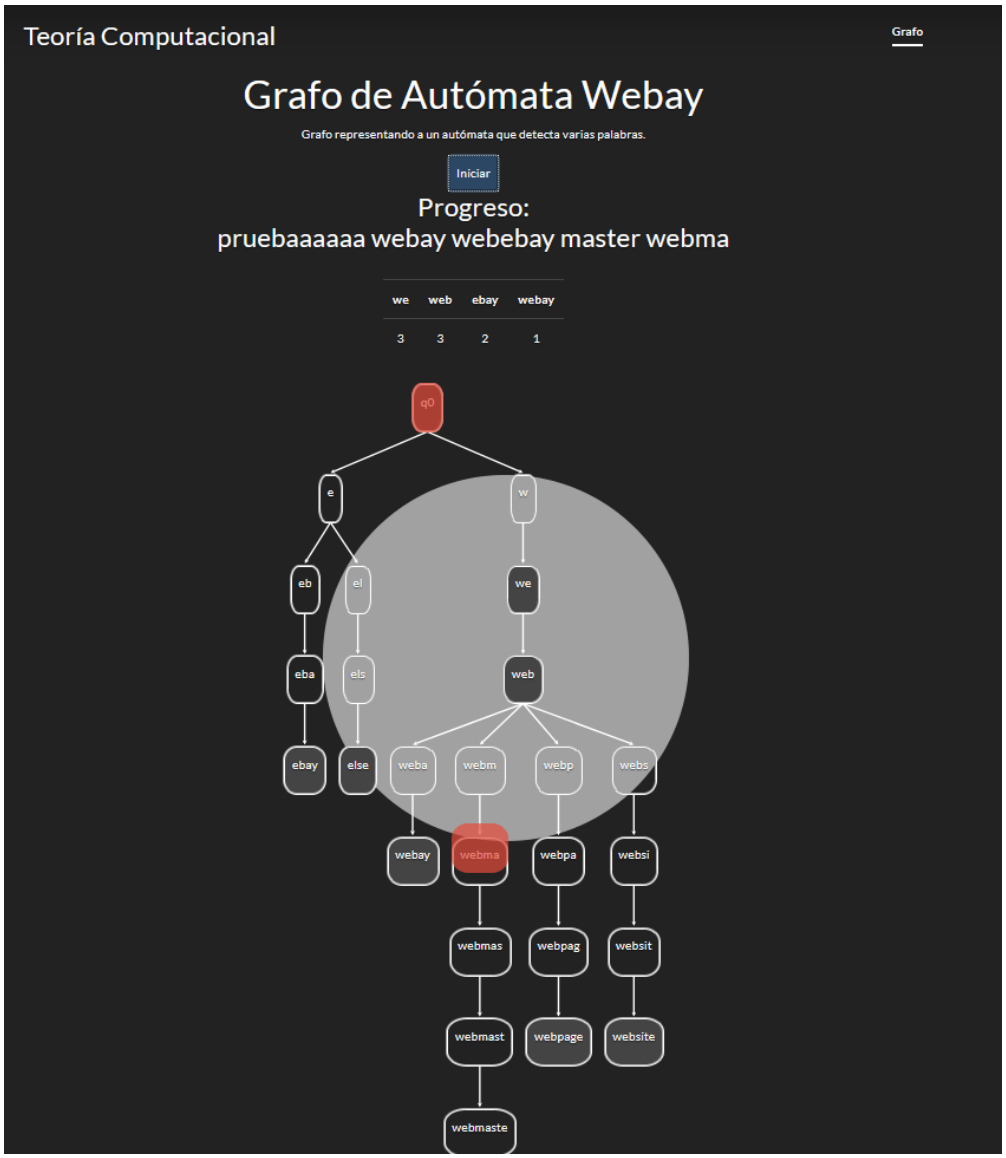

Figura 3: Tabla de Conversión

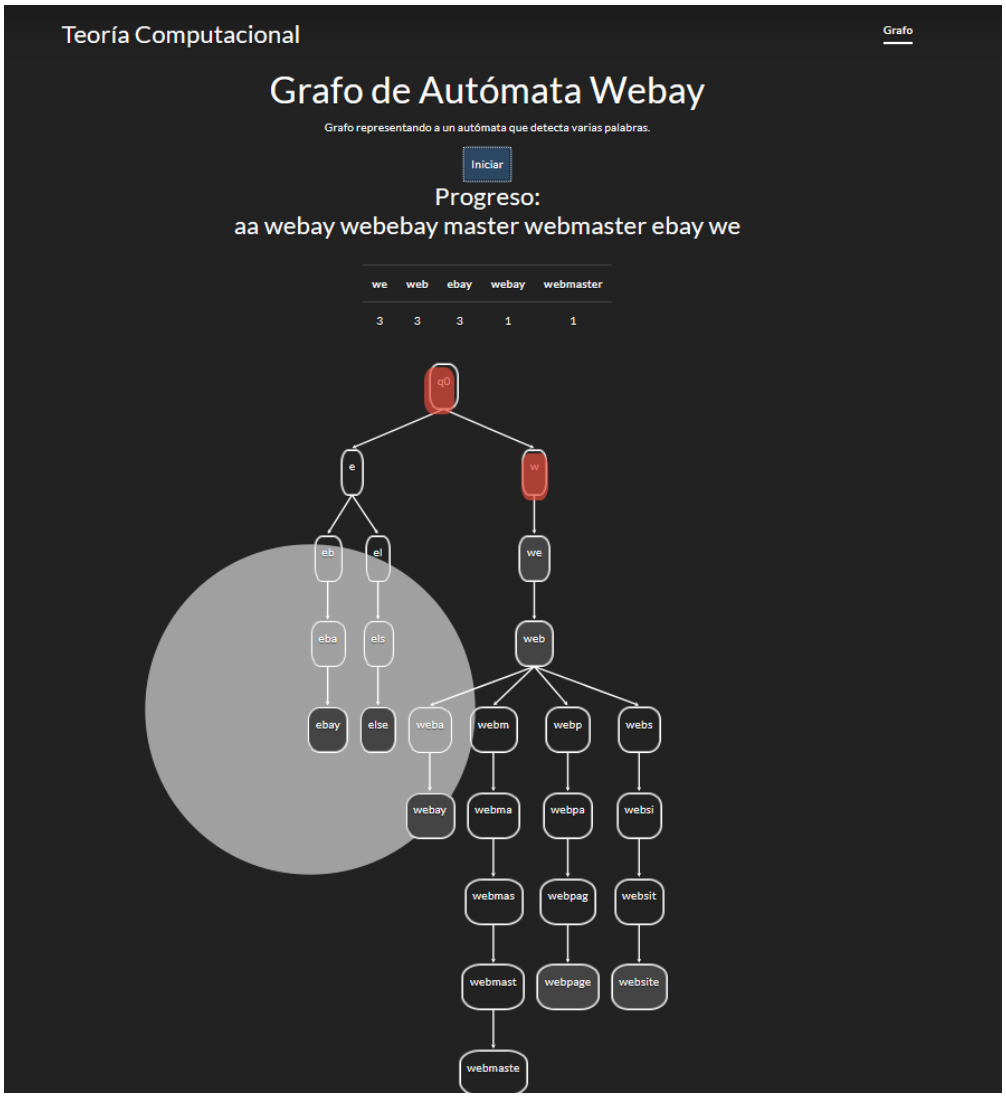

Figura 4: Tabla de Conversión

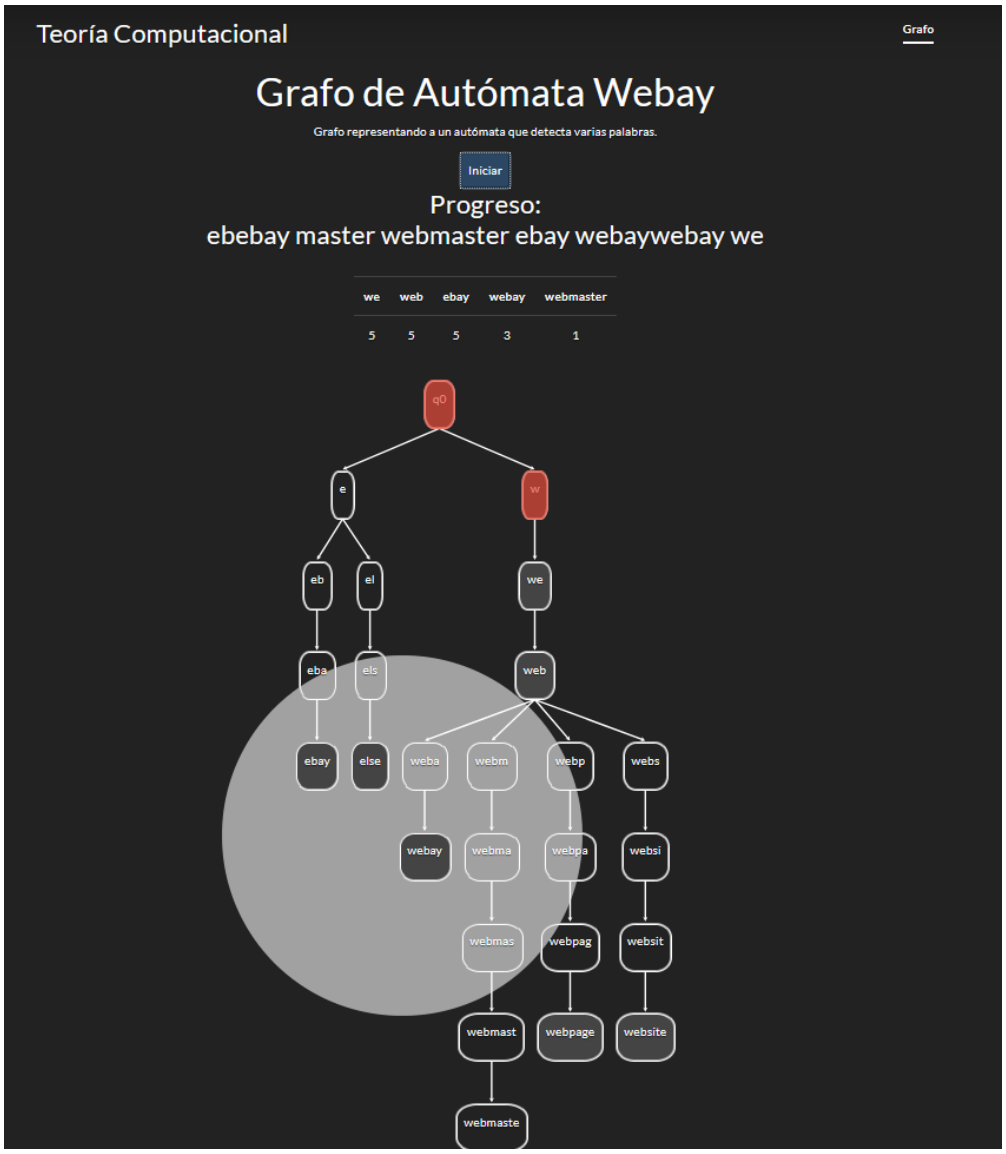

Figura 5: Tabla de Conversión

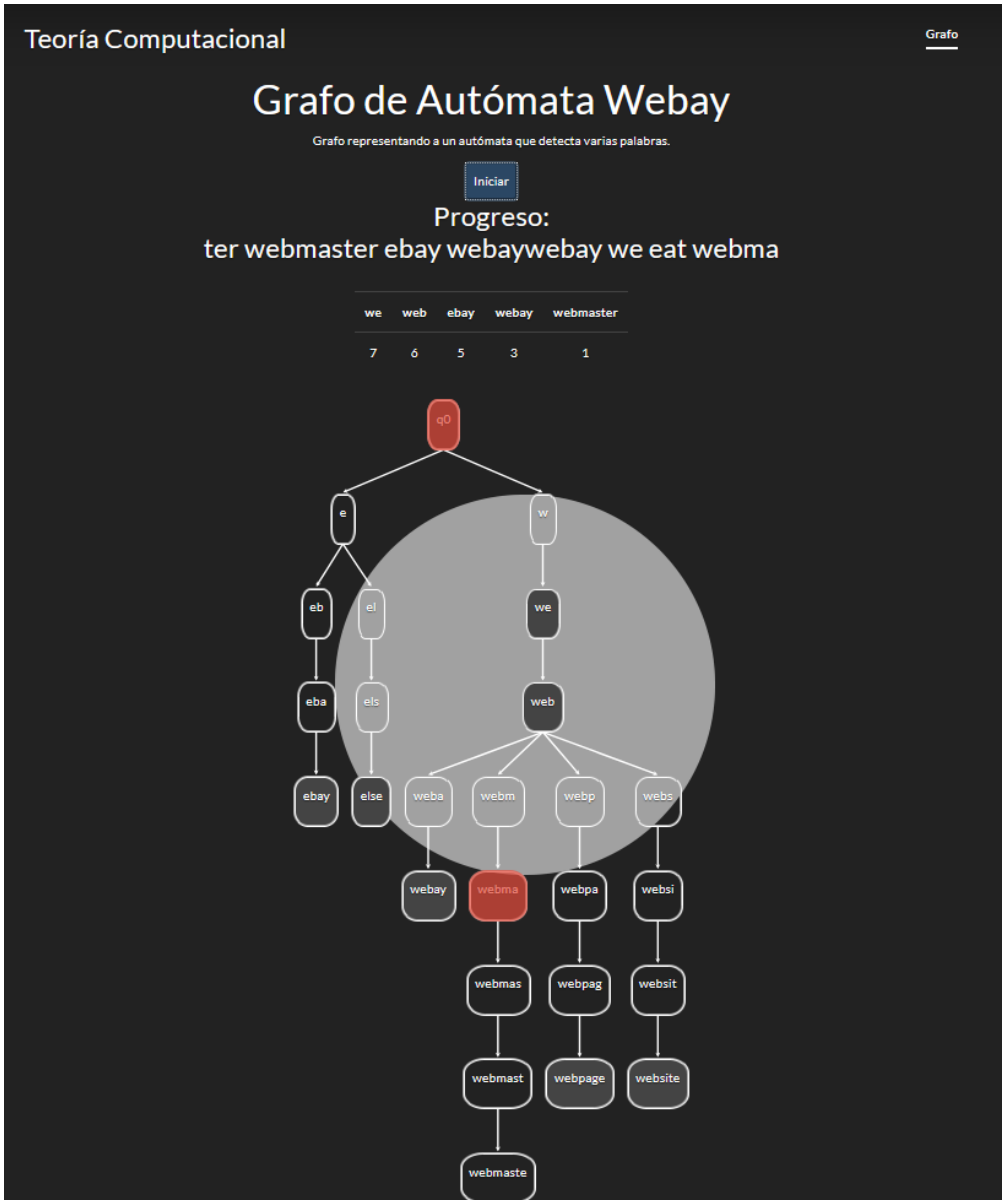

Figura 6: Tabla de Conversión

Autómata finito determinístico: *(Figura 6)*.

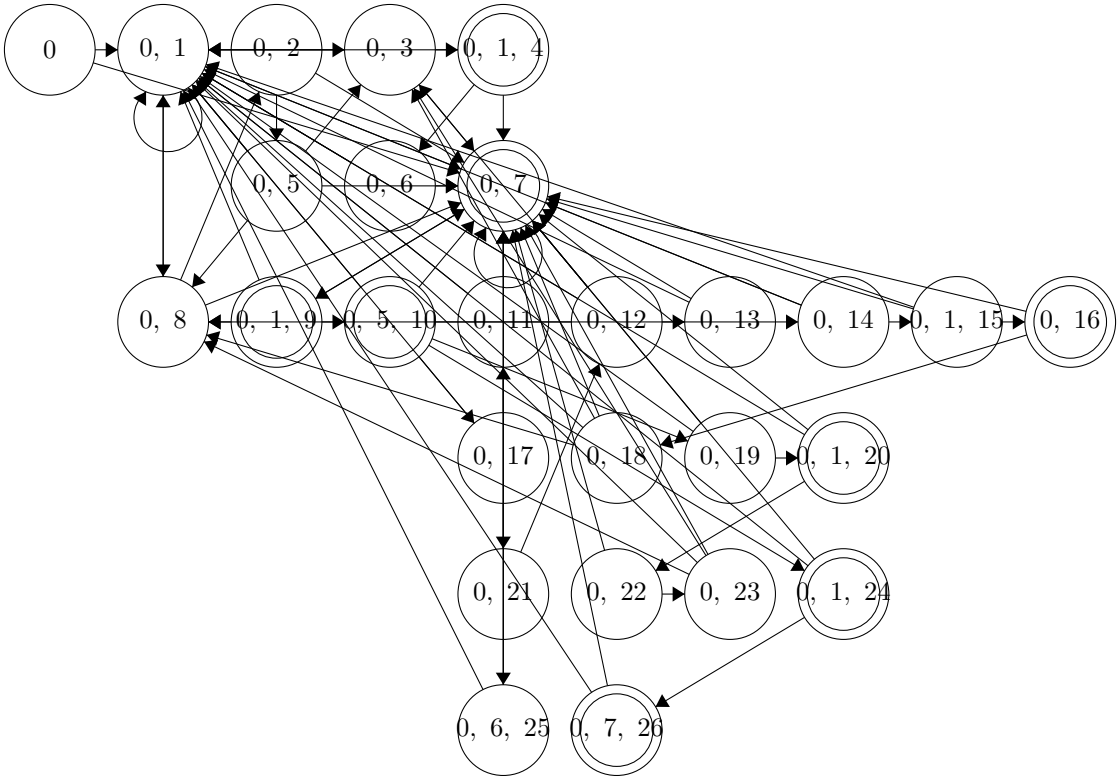

Figura 7: Grafo del Autómata Finito Determinístico