



INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Cómputo

*Práctica #3*

# AUTOMATA

*Axel Treviño Palacios*

2CM5

31 de Octubre de 2020

# 1. Objetivo

Programar el autómata finito determinístico que reconozca todas las palabras reservadas del lenguaje ANSI C.

1. El programa debe de preguntar si quiere calcular otra 'n' o no.
2. El programa deberá de leer un código (archivo) cualquiera.
3. El autómata deberá de identificar cada palabra reservada, contarlas y al final deberá decir cuántas encontró de cada una de ellas.
4. En un archivo imprimir la evaluación del autómata por cada carácter que lea y cambio de estado, es decir, toda la historia del proceso.
5. En otro archivo enumerar y contar cuántas palabras reservadas fueron encontradas.
6. Tener una opción para ver el autómata, es decir, hay que graficarlo.

# 2. Codigo

```
1  const express = require("express");
2  const fs = require("fs");
3  const sanitize = require("sanitize")();
4  const readline = require("readline");
5
6  const app = express();
7  const webPort = 8080;
8  const diccionario = "./diccionario.txt";
9  const archivoFuente = "./archivo.txt";
10 let q0;
11 let JSONTreantTree = {};
12 let counter = {};
13
14 const http = require("http").createServer(app);
15 const io = require("socket.io")(http);
16
17 // Set the view engine to ejs
18 app.set("view engine", "ejs");
19 // Set views path
20 app.set("views", __dirname + "../dist/pages");
21
22 // SERVER SET-UP
23 app.use(express.static("dist/public"));
24
25 // REQUESTS
26 // "static" pages
27 app.get("/", (req, res) => {
28     res.render("index");
29 });
30 app.get("/graph-tree/", (req, res) => {
31     res.end(JSON.stringify(JSONTreantTree));
32 });
33 app.get("/start/", async (req, res) => {
34     res.end(
35         JSON.stringify({
36             start: true,
37         })
38     );
```

```

39         console.log("Starting...");
40         await processFile();
41     });
42     // Get filename and path
43     app.post("/set/path/", (req, res) => {
44         archivoFuente = req.query.filepath;
45         var response = {
46             valid: false,
47             message: "Archivo no encontrado!",
48         };
49
50         // Check if it exists
51         try {
52             if (fs.existsSync(path)) {
53                 response.valid = true;
54                 response.message = "Archivo cargado!";
55             }
56         } catch (err) {}
57
58         res.end(JSON.stringify(response));
59     });
60
61     // WEB SOCKET
62     io.on("connection", (socket) => {
63         var data = {
64             event: "handshake",
65             data: "Hola! C:",
66         };
67
68         console.log("Conexion a socket!");
69         socket.emit("handshake", JSON.stringify(data));
70     });
71     io.sockets.on("result", (data) => {
72         console.log("Result!");
73         console.log(JSON.parse(data));
74     });
75
76     // SERVER LISTEN INIT
77     http.listen(webPort, () => {
78         console.log("Listening on port: " + webPort);
79     });
80
81     // Node Management
82     class QNode {
83         constructor(name, origin, end = false) {
84             this.name = name;
85             this.origin = origin;
86             this.children = {};
87             this.end = end;
88         }
89
90         process(char, ifNot, callback = () => {}) {
91             let next = ifNot;
92
93             for (let [key, val] of Object.entries(this.children)) {
94                 if (key === char) {

```

```

95             next = val;
96             break;
97         }
98     }
99     callback(this, char, next.name);
100     return next;
101 }
102 }
103
104 function generateTree() {
105     console.log("Generating tree from " + diccionario);
106     var lines = fs
107         .readFileSync(diccionario, "utf-8")
108         .replace(/\r/g, "")
109         .split("\n")
110         .filter(Boolean);
111
112     q0 = new QNode("Origen", "\0");
113
114     lines.forEach((line) => {
115         q0 = addNode(q0, line, line);
116     });
117 }
118
119 async function processFile() {
120     const readable = readline.createInterface({
121         input: fs.createReadStream(archivoFuente, { encoding: "utf8" }),
122     });
123     const writeStream = fs.createWriteStream("./proceso.txt");
124     let currentNode = q0;
125     let message = {};
126     let limi;
127     let progreso;
128
129     readable.on("line", async function (line) {
130         line.replace(/\r\n/gi, "");
131         limi = line.length;
132
133         console.log(line);
134         for (let i = 0; i < limi; ++i) {
135             console.log(line[i]);
136             // Node process
137             currentNode = currentNode.process(
138                 line[i],
139                 q0,
140                 (node, char, next) => {
141                     progreso = `${node.name}(${char}) -> ${next}\n`;
142                     console.log(progreso);
143                     writeStream.write(progreso);
144
145                     if (node.end) {
146                         if (typeof counter[node.name] == "undefined")
147                             counter[node.name] = 0;
148                     }
149                     ++counter[node.name];
150                     writeCounter();

```

```

151                                     }
152
153                                     message.process = char;
154                                     message.nodeName = node.name;
155                                     message.nodeEnd = node.end;
156                                     console.log(message);
157
158                                     io.sockets.emit("update", JSON.stringify(message));
159                                     }
160                                     );
161                                     }
162                                     });
163     }
164
165     // Recursively adds a node
166     function addNode(node, word, value) {
167         let length;
168         let nodeName = value.substring(0, value.length - (word.length - 1));
169
170         if (word.length === 1) {
171             node.children[word] = new QNode(value, word, true);
172         } else {
173             // If child does not exist, create it.
174             if (typeof node.children[word[0]] === "undefined") {
175                 node.children[word[0]] = new QNode(nodeName, word.substring(0, 1));
176             }
177
178             // Add children node
179             node.children[word[0]] = addNode(
180                 node.children[word[0]],
181                 word.substring(1),
182                 value
183             );
184         }
185         return node;
186     }
187
188     function writeCounter() {
189         fs.writeFile("./contador.txt", JSON.stringify(counter, null, 4), function() {});
190     }
191
192     function generateGraphTree() {
193         JSONTreantTree.chart = JSON.parse(fs.readFileSync("src/chart.json"));
194
195         JSONTreantTree.nodeStructure = getGraphNode(q0);
196     }
197     function getGraphNode(node) {
198         let result = {};
199         let children = [];
200         result.text = {
201             name: node.name,
202         };
203         if (node.end) {
204             result.HTMLclass = "node-end";
205         }
206     }

```

```

207         for (let [key, val] of Object.entries(node.children)) {
208             children.push(getGraphNode(val));
209         }
210
211         result.children = children;
212         return result;
213     }
214
215     // READ TREE SOURCE FILE
216     generateTree();
217     generateGraphTree();
218     processFile();

```

### 3. Conclusiones

Un autómata es una de las mejores herramientas que tiene un programador cuando existe la posibilidad de que el archivo de entrada sea demasiado grande para cualquier otra alternativa.

### 4. Resultado

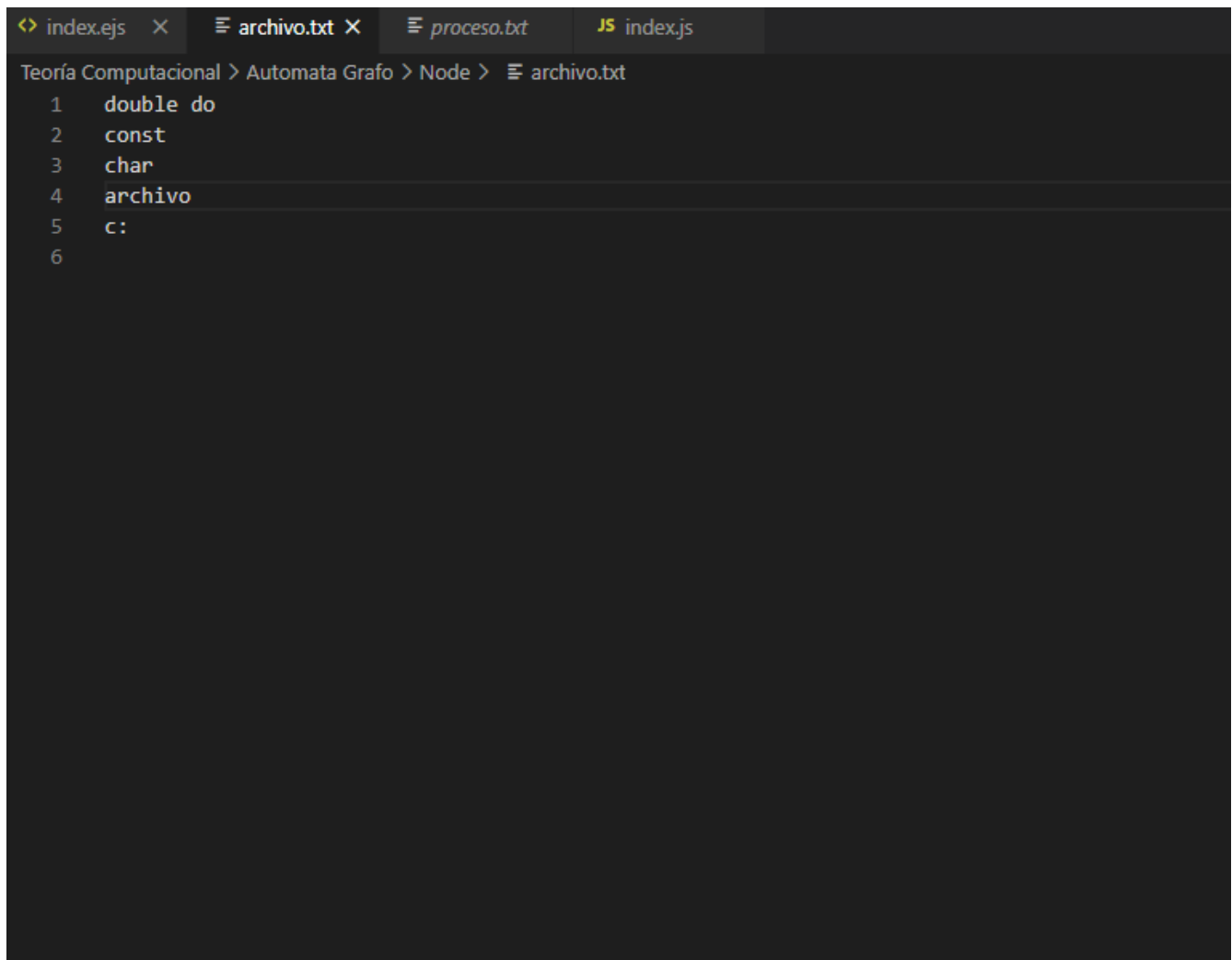
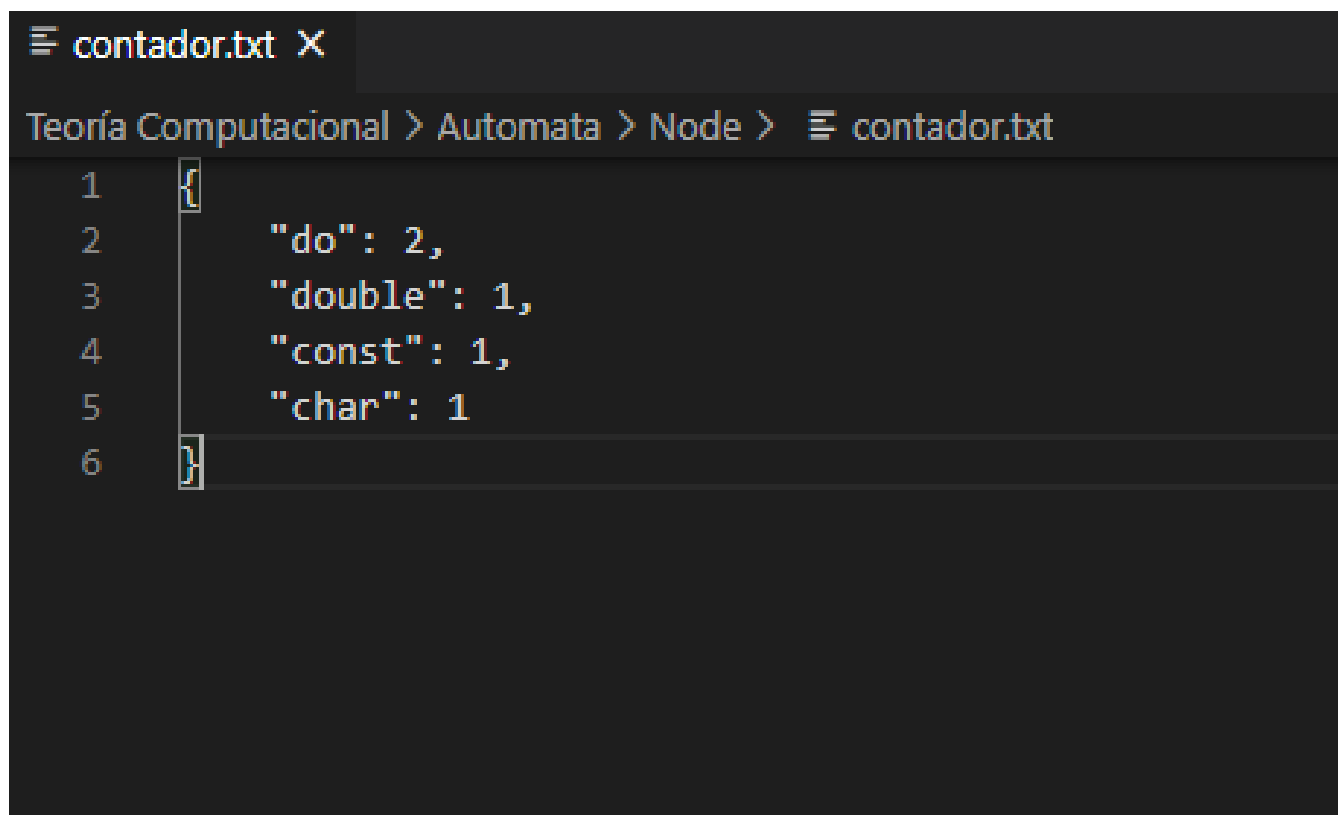


Figura 1: Archivo de entrada prueba



The image shows a code editor window with a dark theme. The title bar at the top reads "contador.txt" with a close button. Below the title bar is a breadcrumb navigation path: "Teoría Computacional > Automata > Node > contador.txt". The main editing area contains a JSON object with four key-value pairs, each on a new line. The lines are numbered 1 through 6 on the left margin. The JSON object is enclosed in curly braces, and the key-value pairs are separated by commas. The keys are in double quotes, and the values are integers.

```
1 {  
2   "do": 2,  
3   "double": 1,  
4   "const": 1,  
5   "char": 1  
6 }
```

Figura 2: Archivo contador

```
Teoría Computacional > Automata Grafo > Node > proceso.txt
1  Origen(d) -> d
2  d(o) -> do
3  do(u) -> dou
4  dou(u) -> Origen
5  Origen(b) -> b
6  b(l) -> Origen
7  Origen(e) -> e
8  e( ) -> Origen
9  Origen(d) -> d
10 d(o) -> do
11 do(c) -> Origen
12 Origen(o) -> Origen
13 Origen(n) -> Origen
14 Origen(s) -> s
15 s(t) -> st
16 st(c) -> Origen
17 Origen(h) -> Origen
18 Origen(a) -> a
19 a(r) -> Origen
20 Origen(a) -> a
21 a(r) -> Origen
22 Origen(c) -> c
23 c(h) -> ch
24 ch(i) -> Origen
25 Origen(v) -> v
26 v(o) -> vo
27 vo(c) -> Origen
28 Origen(:) -> Origen
29
```

Figura 3: Archivo de proceso



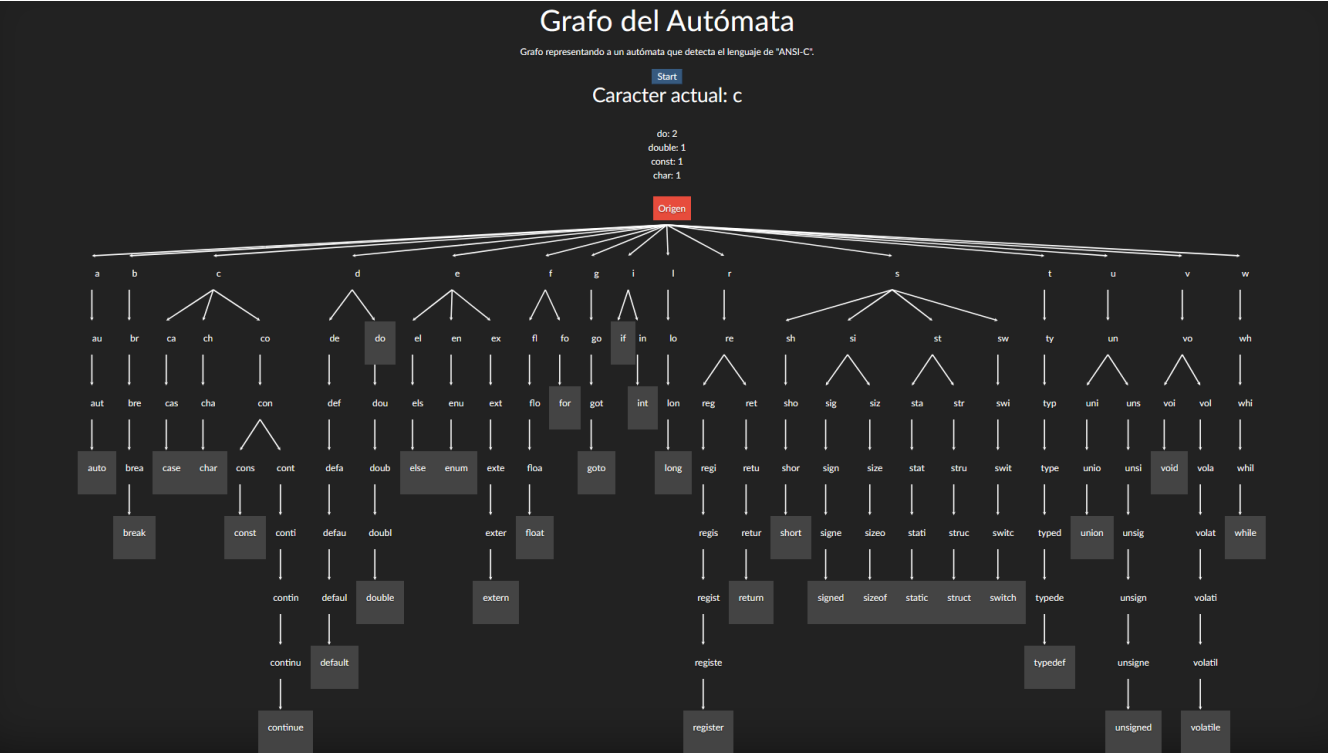


Figura 4: Grafo en el que se representa el proceso