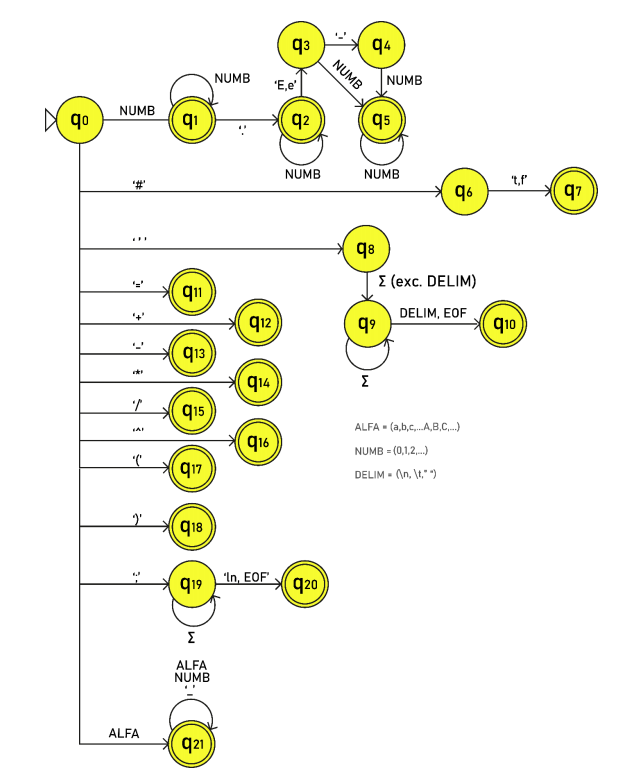
|  |
| --- |
| **Implementación de métodos computacionales**  **TC2037.2**  **Actividad Integradora 6.1:**  **Análisis de herramientas usadas.**    Profesor(a):  Alejandro de Gante    Abraham Mendoza Pérez               A01274857  Luis Alonso Martínez García          A01636255  Aldo Alejandro Degollado Padilla A01638391      Viernes 28 de abril de 2021 |

Conforme se iba estudiando los distintos temas de la materia de Implementación de métodos computacionales (TC2037), se nos presentó la primera parte de lo que sería la situación problema, en la cual, por medio de una las aplicaciones que tienen los autómatas finitos determinísticos, nuestro objetivo como equipo era el realizar un código capaz de identificar y regresar el tipo de tokens al tener como entrada un archivo de texto de nuestra elección, donde posteriormente este proyecto tendría modificaciones para que ahora se pudieran detectar las categorías léxicas, al igual que se realizaría la implementación de HTML+CSS para poder resaltar el léxico detectado por el programa.

La resolución de toda esta situación problema no sólo conllevó el uso de programación, sino que de igual manera el uso de distintas herramientas que iban desde diseño, trabajo colaborativo, IDE e incluso la aplicación de conceptos en alguna de las herramientas mencionadas anteriormente fueron de suma importancia para la conclusión exitosa del proyecto. Por ello en este ensayo se realizará un análisis a detalle sobre como estás se fueron implementando a lo largo del desarrollo de la solución, las alternativas disponibles que pudieran utilizarse y otros criterios de evaluación para la misma.

Desde que se planteó la situación, el equipo al estar analizando de qué manera podríamos resolver esta, fue que con los primeros temas vistos en clase uno de los más llamativos y fáciles de aplicar para nosotros fue los autómatas finitos deterministas ya que desde un inicio el código que íbamos a realizar principalmente se iba a basar en el uso de switch statement, ya que dependiendo del carácter que se detectará, es que este iba determinando poco a poco la clasificación de token (categoría léxica posteriormente) de acuerdo a las restricciones dadas y este funcionamiento como tal se asemeja en su mayoría al de un autómata, ya que en este lenguaje principalmente se ve formado por distintos procesos los cuales se van recorriendo de acuerdo a la entrada de un símbolo donde para poder realizar está transacción es necesario que el símbolo se encuentre presente dentro del siguiente proceso. De igual manera estos pueden volver recorrer el mismo el proceso en caso de ser necesario y al cumplirse de manera exitosa un camino de estos procesos, habrá un estado final en el cual se dice que autómata acepta de manera correcta los símbolos ingresados. Así fue como finalmente llegamos a plantear nuestra lógica del funcionamiento del programa en un DFA.



De igual manera no sólo esta era la única forma de tener un guía base para el desarrollo del código, sino que están presentes las expresiones regulares(también conocidas como regex), otro de los temas revisados durante la materia. Estos principalmente consisten en una cadena de caracteres utilizadas para encontrar patrones o combinaciones dentro de otras esto conforme a cierto uso de delimitadores y algunas reglas de sintaxis. Estos cuentan metacaracteres los cuales se pueden ver como condiciones que crea el mismo usuario en caso de requerirlo.

Como tal estos si pueden representar una opción alternativa bastante llamativa, ya que si plantaremos un nuevo punto de vista de nuestro código donde ahora nos enfocáramos en realizar la clasificación léxica de acuerdo a que los patrones de estos coincidieran con la entrada que se está dando en el código se podría realizar sin ningún inconveniente e incluso programas como Java cuentan con un set de funciones tales como exec() y test() de RegExp, con match(), matchAll(), replace() entre otras las cuales podrían ser de gran ayuda pero como el equipo no tenía una idea muy clara de las expresiones, ninguno había utilizado Java, a profundidad anteriormente, y posiblemente nos hubiera mucho más tiempo adaptarnos con estos, lo mejor fue hacer uso del autómata aunque no se desmerita a las expresiones regulares, ya que una vez aprendiendo a utilizar estás, se pueden aplicar en bastante ámbitos de la informática.

En cuanto al trabajo colaborativo, para un mejor manejo de código, e igualmente de documentos, nosotros utilizamos GitHub por ser requisito del profesor, además de su popularidad, y que siempre es la primera opción para soluciones de este tipo. Sin embargo, existen otras alternativas como Gitlab, que ofrece numerosas y útiles características en su DVCS, como, por ejemplo, un proyecto wiki integrado y una página web de proyecto. Las continuas capacidades de integración de GitLab automatizan el análisis y la entrega del código, lo que permite ahorrar tiempo en la fase de prueba. Con un visor de código, pull requests y un práctico método para solucionar conflictos, GitLab permite acceder a todos los aspectos más importantes de tu proyecto. En documentos, consideramos que una alternativa que sería más eficiente es un procesador de texto en línea como Google Docs, pues facilita la edición observando los cambios realizados por los otros colaboradores en tiempo real.

En nuestro caso, por ser un lenguaje en el cuál hemos tenido más práctica, utilizamos C++, dónde se realizó el algoritmo para el resaltador de sintaxis, que después de realizar el análisis, generaba un archivo HTML, con los estilos que determinamos en otro archivo CSS. Para ello, otra solución que podría mejorar este proceso sería realizarlo en un lenguaje de programación como Python, donde podemos utilizar algún motor de plantillas como django template, jinja2 o genshi.

Por a parte del uso de herramientas UML, app.diagrams.net fue la página la cual equipo sólo utilizó al momento de diseñar el diagrama de nuestro autómata finito para la solución de la situación, aunque este no es necesariamente para hacer demostraciones de diagramas de este tipo, ya que app diagrams ofrece una gran variedad de tipos de diagramas que pueden utilizarse para demostrar el funcionamiento más a detalle de un código que esté en desarrollo o incluso ya haya sido finalizado, aspectos más técnicos que sean necesarios mostrarse al equipo de trabajo o viendo desde un lado más profesional a un cliente y se nos ofrece una amplia gama de opciones en caso de necesitar descargar alguno de los trabajos realizados o incluso poder guardar estos en la nube como Google Drive o One Drive en caso de requerirlo. Sin embargo, está no es la única herramienta de diagramas UML disponible en la red ya que existen otras herramientas alternativas entre las cuales podemos encontrar a LucidChart. Este al igual que app.diagrams, nos ofrece una amplia variedad de distintos diagramas, aunque en este caso, está herramienta se ve apoyada por distintos plugins de integración de plataformas tales como Java, Slack, Google Drive y Microsoft Teams donde todo el sistema del interfaz y como funciona va más dirigido hacia gente que no se dedique como tal para esto o vaya iniciando ya que por la parte diagrams.net, se nos especifica que vas orientada a personas expertas como desarrolladores, administrador de redes, diseñadores, entre otros.

Otra herramienta la cual nos fue de gran utilidad para probar si el autómata finito que habíamos creado sería capaz de cubrir la situación problema planteada fue JFlap, la cual nos ofrece opciones como diseñar y usar expresiones regulares, gramáticas, autómatas de pilas y finitos. Esta herramienta funciona con Java por lo cual es necesario contar un JDK para su uso. Al usarla tenemos un menú con muchas opciones y obviamente en nuestro caso optamos por el autómata finito donde al usarlo, podemos ir creando los distintos procesos que necesitamos al igual que los símbolos que son necesarios para las transiciones y es funcional ya que nos permite probar con entradas del usuario y en el programa se nos va mostrando como se va recorriendo los distintos procesos y la finalización de este.

Aunque el equipo durante las pruebas del DFA, encontró una herramienta alternativa conocida como Jsflap, una herramienta para la construcción de NFA Y DFA por medio de la Web y algo interesante es que está se encuentra basada en JFlap por lo cual pensamos que sería una buena opción en caso de tener algún problema con JFlap, aunque desde la perspectiva de todos los integrantes Jsflap es inferior a JFlap en aspectos como opciones de diseño ya que por parte de Jsflap solo podemos crear un autómata finito o una máquina de Turing y que no cuenta con una versión descargable pero fuera de eso, si se busca crear un diseño rápido y ver si es funcional Jsflap cubre esos puntos ya que este de igual manera acepta entradas del usuario que validen los procesos del autómata por lo cual es una herramienta alternativa que pudo llegar a cubrir las necesidades que se nos presentarán en el desarrollo del proyecto.

REFERENCIAS

<https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/alternativas-a-github/>

<https://conocimientolibre.mx/que-es-diagrams-net/#:~:text=Diagrams.net%20es%20una%20aplicación,para%20Windows%2C%20Linux%20y%20macOS>.

<https://www.capterra.es/compare/166985/146136/draw-io/vs/lucidchart>

http://www.jflap.org