INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

GARZÓN DOMÍNGUEZ GERARDO ISMAEL

2CV13

TEORÍA COMPUTACIONAL

PRÁCTICA 2:
LENGUAJES

INTRODUCCIÓN:

Para definir un lenguaje, se puede comenzar desde la idea de un alfabeto que se denotará como Σ. La operación Σ^*, denota el conjunto de todas las cadenas sobre el alfabeto Σ y que son cerradas Σ^* mediante la operación de concatenación, notando que el conjunto Σ^* incluye el elemento identidad. Tras ese razonamiento, de manera muy general, se puede definir un lenguaje L sobre un alfabeto Σ, tal que $L \subset \Sigma^*$ (subconjunto de) [1].
Como se puede observar, un lenguaje está definido sobre la teoría de conjuntos, por lo cual las operaciones de unión, intersección, diferencia, etc. Ya están bien definidas.

De manera informal se puede definir la concatenación entre dos lenguajes L_1, L_2 como una operación binaria, asociativa y no conmutativa de la que se obtiene un conjunto que contiene la concatenación de cada cadena del primer lenguaje con cada cadena del segundo siempre y cuando esta concatenación no de resultados repetidos (esto es una manera más práctica de pensarlo, y es útil desde la perspectiva de implementación de estos conceptos en una computadora).
De una forma estricta, la concatenación entre dos lenguajes L_1, L_2, se define de la siguiente manera:

$$L_1 L_2 = \{xy : x \in L_1, y \in L_2\}$$

A partir de la concatenación de lenguajes se puede definir la potencia de un lenguaje, de tal manera que L^n es la concatenación de L, consigo mismo n veces, con los casos especiales en los que: $L^0 = \{\lambda\}$ y $L^1 = L$ [1].
La inversa de un lenguaje denotada como L^{-1} se define como el conjunto de las cadenas de L invertidas, de manera estricta: $L^R = \{w^R : w \in L\}$ [1].
Por lo tanto, un lenguaje $L^{n\leq 1}$, es simplemente la inversa del lenguaje concatenada consigo mismo n veces.

Por ultimo, el cierre de Kleene sobre un conjunto de cadenas L está definido y denotado como: $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$ Que es la unión de todas las potencias de un lenguaje.
El cierre positivo está definido y denotado como: $L^* = L^1 \cup L^2 \cup L^3 \cup \ldots$[1] Que es la unión de todas las potencias de un lenguaje excepto la potencia 0, por lo cual, se excluye la cadena vacía si y solo si el lenguaje L no la contiene, o sea si $\lambda \notin L$.

DESARROLLO E IMPLEMENTACIÓN

El aspecto que notar en esta práctica es que estamos trabajando con conjuntos de cadenas. Los conjuntos por definición no pueden tener elementos repetidos, y este fue el principal problema a resolver en el diseño e implementación del tipo abstracto de dato “conjunto de cadenas” (strset_t en el código.)
Inicialmente implemente este tipo como un arreglo dinámico con apuntadores a objetos cadena, Sin embargo, las pruebas realizadas mostraron tiempos largos para operaciones como cierre de Kleene y potencias. Al analizar las operaciones que se realizar para insertar elementos, se puede notar que conforme el conjunto crece, se debe realizar una búsqueda lineal para comprobar que no haya elementos repetidos, para conjuntos lo suficientemente grandes, se vuelve algo lento e innecesario el tener que realizar una búsqueda lineal por cada inserción, así que para este caso, una tabla hash es mas conveniente, en promedio la búsqueda de elementos toma tiempo constante, y lineal en el peor de los casos, en general depende de la función hash utilizada [2]. Incluso si hay varias colisiones dentro de una lista, sigue siendo mejor que iterar y comparar con cada uno de los elementos ya insertados.
La función hash utilizada para las cadenas es una variante inspirada en “One-at-a-Time Hash” [3] La única diferencia con la función original es que no itera sobre toda la cadena sino un numero de veces calculado a partir de la longitud de la cadena. En caso de que haya colisiones la tabla crece y los elementos se re-mapean a un nuevo índice dentro de la tabla.
Tras varias pruebas se pudo observar que, para conjuntos pequeños, la tabla es un poco más lenta que una lista enlazada o arreglo, sin embargo, para conjuntos mas grandes, se nota que es mas eficiente.

```
chell@bit pr2_good % ./t
A = {"cde", "bc", "a", }
Elements 88573, Buckets 1/98304
38.7901s
```

Figura 1. Tiempo tomado para computar Kleene estrella del conjunto {a, bc, cde} con limite de 10, usando una lista enlazada

```
chell@bit pr2_good % ./t
A = {"a", "cde", "bc", }
Elements 88573, Buckets 12275/12288
1.2627s
```

Figura 2. Tiempo tomado para computar Kleene estrella del conjunto {a, bc, cde} con limite de 10, usando una tabla hash

FUNCIONAMIENTO

En esta sección se mostrará el funcionamiento de la función “potencia” usando el lenguaje de prueba dado {a, ab, cde}

```
A = {"a", "cde", "bc", }

|A(-2)| = 9
A(-2) = {"cba", "cbcb", "edcedc", "acb", "aedc", "edca", "cbedc", "aa", "edccb", }
```

Figura 3. Conjunto de prueba, elevado a la -2

```
A = {"a", "cde", "bc", }

|A(-5)| = 243
```

Figura 4. Conjunto de prueba elevado a la -5, el conjunto no se imprime por su tamaño, pero la cardinalidad del conjunto se muestra

```
A = {"a", "cde", "bc", }

|A(0)| = 1
A(0) = {"", }
```

Figura 5. Conjunto prueba “a” elevado a la 0, se obtiene el conjunto con la cadena vacía.

```
A = {"a", "cde", "bc", }

|A(3)| = 27
A(3) = {"bccdecde", "abca", "bcacde", "bccdebc", "cdecdecde", "cdebca", "acdebc", "cdecdea", "bcbca"
, "aaa", "acdea", "aabc", "cdebccde", "acdecde", "aacde", "bcbccde", "bcbcbc", "cdeabc", "cdeaa", "b
caa", "cdebcbc", "bccdea", "cdecdebc", "cdeacde", "bcabc", "abccde", "abcbc", }
```

Figura 6. Conjunto prueba elevado a la 3, su cardinalidad es 27.

```
A = {"a", "cde", "bc", }

|A(11)| = 177147
```

Figura 7 Conjunto de prueba elevado a la 11, no se imprime por su tamaño, su cardinalidad se muestra

CONCLUSION

Fue bastante útil implementar los conceptos de lenguajes, ya que estos resultan algo abstractos por estar definidos sobre teoría de conjuntos (que es parte de la base de lógica matemática). Trasladarlos a un lenguaje de programación hace que se note la complejidad que tienen las matemáticas en general por que es que las computadoras son bastante útiles para este tipo de cosas. Además, se nota el uso de ciertos conocimientos previamente adquiridos, como las matemáticas discretas, pues en este caso fueron de utilidad para simplificar un poco la implementación de ciertas funciones como estrella de Kleene y Kleene mas. Fue bastante interesante implementar el tipo abstracto de dato para conjuntos de cadenas y sus funciones desde las más elementales como la inserción, y definir otras operaciones en base a esta, como la unión, intersección y diferencia, en el caso de potencias y concatenaciones uno de los retos fue el hacer un uso mas o menos razonable sobre la memoria. A pesar de que los conjuntos son algo bastante abstracto, pasan a ser algo un poco más concreto si lo pensamos desde el punto de vista computacional, dado que este caso, se puede pensar en ellos como algo en términos de memoria y algoritmos. Otro aspecto de utilidad que puedo rescatar de esta práctica es que me incentivó a investigar un poco más sobre como realizar una implementación más conveniente para la resolución del problema, en este caso fue, implementar los conjuntos como tablas hash para obtener mejor provecho de la computadora cuando esta opere sobre estas estructuras.

Referencias:

[1] P. Linz, *An Introduction to Formal Languages and Automata.* 6ta ed, California: JONES & BARTLETT LEARNING, pp 27-30.

[2] Paul E. Black, "hash table", en *Dictionary of Algorithms and Data Structures* [en linea], Paul E. Black, ed. 12 February 2019. (accedido 22-Mar-21) Disponible en: https://www.nist.gov/dads/HTML/hashtab.html

[2] B. Jenkins en *Hash functions.* [en linea] Septiembre 1997. (accedido 22-Mar-21) Disponible en: https://www.drdobbs.com/database/algorithm-alley/184410284