Proyecto

Teoría de la computación

3 de mayo de 2022

Resumen

Este es un proyecto grupal para el curso de Teoría de la Computación, vale el 20 % de la nota final. Puede ser hecho por grupos de máximo 3 personas.

1. Búsqueda en texto

Una aplicación importante de la Teoría de Autómatas es la búsqueda de palabras clave en la web. Por ejemplo, al buscar en google ciertas palabras clave, quisieramos encontrar páginas web que contengan por lo menos una de estas palabras. Visto de otra forma, quisieramos evaluar si un cierto documento tiene por lo menos alguna de las palabras clave dadas. Formalmente, podemos describir dicho problema de la siguiente manera.

Problema SUBSTRING. Dadas una cadena s y un conjunto de cadenas T sobre un alfabeto Σ, decidir (responder sí o no), si es que alguna cadena en T es una subcadena de s.

Por ejemplo, si $s = abcdefghij$, y $T = \{efg, iii, kjk\}$ la respuesta es SÍ. Sin embargo, si $T = \{hijk\}$, la respuesta es NO
El problema SUBSTRING es resoluble usando autómatas. Para esto, basta construir un AFN que, esperando en un estado q_1, intenta adivinar no deterministicamente si la cadena recibida está en T. Un ejemplo se muestra en la Figura 1, en la cual $T = \{web, ebay\}$.

Pregunta 1. Construir un AFN que permita resolver el problema SUBSTRING.
Entrada del algoritmo: Un alfabeto Σ, y un conjunto de cadenas T sobre Σ.
Salida del algoritmo: Un AFN que permita resolver SUBTRING con parámetros Σ, T.

Algunas especificaciones de implementación para la pregunta anterior. Tenga en cuenta que trabajaremos con entrada estándar.

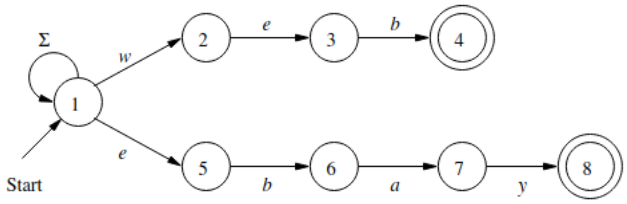

Figura 1: Tomada del libro de Hopcroft.

- Sobre el input: en la primera línea se leerá una cadena, donde Σ está conformado por todos los caracteres de dicha cadena. En la segunda línea se leerá un número t que indica el número de cadenas en T. En las siguientes t líneas se tienen los elementos de T.
- Sobre el output: Deberá ser devuelta una estructura especial para guardar AFN. Esta estructura deberá usar listas de adyacencia, tal y como se hace al almacenar grafos.

Luego de resolver la pregunta 1, tendremos un AFN que permite resolver el problema SUBSTRING. A partir de acá tenemos dos opciones. La primera consiste en simular cada uno de los movimientos del AFN. La segunda consiste en transformar el AFN a un AFD y hacer la simulación en este AFD resultante.

Una implementación interesante que simula el AFN consiste en adaptar una búsqueda en largura en grafos (BFS). Para ello, usaremos una cola que mantiene en un determinado momento, todos los estados a los cuales puedo llegar luego de leer cierto número de caracteres. A partir de estos estados, los retiramos uno a uno de la cola haciéndolos transicionar con el siguiente caracter. Luego de esto, tenemos un nuevo conjunto de estados en la cola.

Pregunta 2. Resolver el problema SUBSTRING usando la adaptación de BFS sobre un AFN (ver párrafo anterior).
Entrada del algoritmo: Un alfabeto Σ, una cadena s sobre Σ, y un conjunto de cadenas T sobre Σ.
Salida del algoritmo: "SI", si el problema SUBSTRING devuelve verdadero, y "NO", en caso contrario.

La especificación de input para la Pregunta 2 es la misma que para la pregunta anterior. Además, es obligatorio usar la Pregunta 1 como subrutina para la Pregunta 2.

La siguiente manera de resolver nuestro problema es transformando el AFN a AFD usando el algoritmo visto en clase. Una vez que se tiene un AFD, es muy facil saber si la cadena s es aceptada o no por el AFD.

Pregunta 3. Resolver el problema SUBSTRING usando el algoritmo visto en clase que transforma AFN en AFD (ver párrafo anterior).
Entrada del algoritmo: Un alfabeto Σ, una cadena s sobre Σ, y un conjunto de cadenas T sobre Σ.

Salida del algoritmo: "SI", si el problema SUBSTRING devuelve verdadero, y "NO", en caso contrario.

Para la pregunta 3 usaremos la misma manera de leer el input que en las preguntas anteriores. Además, para guardar el AFD resultante de la transformación, basta usar la misma estructura de datos que para nuestro AFN (si desea, también es posible usar una estructura más simplificada).

Note que el AFN transformado es un tanto particular, por lo tanto este último algoritmo se puede simplificar. Para este punto, es importante consultar la sección 2.4.3 del libro de Hopcroft. Resumidamente, se tienen la siguientes reglas para generar los nuevos estados.

- Si q_0 es el estado inicial del AFN, entonces creamos $\{q_0\}$ como siendo el estado inicial del AFD.
- Sea p es un estado del AFN. Sea s la cadena generada a partir de los caracteres existentes en el camino simple desde q_0 hasta p en el AFN. Creamos un estado $\{q_0, p\} \cup A$, donde A es el conjunto de estados en el AFN que son alcanzables desde q_0 a través de un sufijo de s.

Pregunta 4. Resolver el problema SUBSTRING usando la transformación especial de AFN a AFD (ver párrafo anterior).
Entrada del algoritmo: Un alfabeto Σ, una cadena s sobre Σ, y un conjunto de cadenas T sobre Σ.
Salida del algoritmo: "SI", si el problema SUBSTRING devuelve verdadero, y "NO", en caso contrario.

Note que cada pregunta tiene un tiempo de ejecución distinto en función de los parámetros de entrada. Para cada una de ellas, debe analizar y justificar cual es este orden en notación O-grande.

2. Test de lenguaje vacío en G.I.C.

Un problema interesante en Teoría de Lenguajes Formales es determinar si una Gramática Independiente del Contexto (GIC) genera por lo menos una cadena. Más formalente,

Problema Test-Vacio. Dada una GIC G, decidir (responder sí o no), si es que $L(G) = \emptyset$.

Por ejemplo, si G es la siguiente GIC,

S –> A|BA
A –> S
B –> b ,

entonces $L(G) = \emptyset$. Existen muchos otros ejemplos, como los vistos en clase en los cuales $L(G) \neq \emptyset$ para alguna otra G.I.C G.

Este problema puede ser resoluble en tiempo polinomial. Sea n el tamaño total de la G.I.C (en el ejemplo anterior, la G.I.C. tiene tamaño 12). Consideraremos dos algoritmos para resolverlo. Para entender mejor estos algoritmos, recomendamos leer la sección 7.1.2 y 7.4.3 del libro de Hopcroft. Adoptaremos también la terminología de dicho libro. Para ello, es importante diferenciar los términos *variables*, *terminales* y *símbolos*. Además, decimos que un símbolo es *generador* si es que existe una secuencia de derivaciones (posiblemente una secuencia nula) que permite llegar de dicho símbolo a alguna cadena consistente en terminales.

El primer algoritmo es un algoritmo $O(n^2)$. Se basa en la siguiente observación. Es claro que un símbolo terminal es generador, ya que se genera si mismo siguiendo cero pasos. Además, si tenemos una regla $A \to \alpha$ y todos los símbolos en α son generadores, entonces la variable A también es generadora.

De esta manera, podemos diseñar un algoritmo $O(n^2)$ para nuestro problema. Durante el algoritmo mantenemos información de los símbolos generadores hasta el momento. Al inicio los símbolos generadores son los terminales. En cada iteración recorremos todas las reglas de la gramática y verificamos si el lado derecho de cada regla tiene solo símbolos generadores, si es así, adicionamos la variable correspondiente a nuestro conjunto de generadores. Al haber como máximo n iteraciones, el algoritmo tiene la complejidad deseada.

Pregunta 5. Resolver el problema Test-Vacio usando el algoritmo $O(n^2)$ descrito anteriormente.
Entrada del algoritmo: Una G.I.C G de tamaño n.
Salida del algoritmo: "SI", si el problema Test-Vacio devuelve verdadero, y "NO", en caso contrario.

Podemos mejorar el algoritmo anterior usando un algoritmo $O(n)$. La idea del algoritmo es mantener estructuras de datos que nos permitan acceder rapidamente a la información. Este algoritmo aparece en la sección 7.4.3 del libro de Hopcroft. La idea es llevar un conteo, para cada regla, de cuántos símbolos en su lado derecho han sido descubiertos que son generadores. Si en algún momento sabemos que todos los símbolos lo han sido, entonces adicionamos la variable del lado izquierdo de la regla a una cola. Para procesar un símbolo de la cola, recorremos, mediane una lista enlazada, las posiciones en las cuales se encuentra dicho símbolo en los lados derechos, actualizando el número de símbolos que ya son generadores. El tiempo total es $O(n)$, ya que en cada iteración se tiene un tiempo de ejecución proporcional al número de posiciones en las cuales ocurre dicho símbolo

Pregunta 6. Resolver el problema Test-Vacio usando el algoritmo $O(n)$ descrito anteriormente.
Entrada del algoritmo: Una G.I.C G de tamaño n.
Salida del algoritmo: "SI", si el problema Test-Vacio devuelve verdadero, y "NO", en caso contrario.

Algunas consideraciones para la implementación de las preguntas 5 y 6. Todo nuestro trabajo debera ser ser hecho por entrada y salida estandar. Sobre el input:

- En la primera línea se leerá una cadena en letras minúsculas, donde el conjunto de terminales está conformado por todos los caracteres de dicha cadena.
- En la segunda línea se leerá una cadena en letras mayúsculas, donde el conjunto de variables está conformado por todos los caracteres de dicha cadena.
- En la tercera línea se tiene el número de reglas, r. En las siguientes r líneas se tendrá la descripción de cada una de las reglas. Esta consiste en una variable y una cadena separadas por un espacio en blanco.

Tenga en cuenta que el tiempo gastado para inicializar y procesar la gramática leída, deben tener complejidad $O(n)$. Deberá usar para ello listas enlazadas donde sea necesario. Tenga en cuenta también que no puede usar ninguna estructura de datos ya implementada en STL.

3. Consideraciones sobre la implementación

- Usar solo C o C++
- No puede usar STL ni librerias con estructuras de datos ya implementadas.
- Debe manejar la memoria de manera dinámica
- Toda entrada y salida es estandar

4. Experimentación numérica

Debe generar inputs aleatórios para comparar sus algoritmos. Debe mostrar en la práctica que los algoritmos se comportan según el tiempo de ejecución previsto. Para ello deberá mostrar tablas y gráficas que muestran la variación de los tiempos de ejecución según los tamaños de sus inputs generados.

5. Monografía

Deberá contener las siguientes secciones y debe ser escrito en LaTeX

1. Introducción y definición del problema (debe incluir bibliografía).
2. Pseudocódigo de los algoritmos que resuelven el problema (vea https://en.wikibooks.org/wiki/LaTeX/Algorithms). Para cada algoritmo, breve explicación de como funciona y del análisis del tiempo de ejecución del mismo.
3. Código de los algoritmos implementados (vea https://en.wikibooks.org/wiki/LaTeX/Source_Code_Listings).
4. Experimentación numérica

6. Fechas de entrega

Tendremos dos fechas de entrega. Se revisará en clase el código de cada grupo individualmente.

30/05 y 01/06

- Preguntas 1, 2 y 3.
- Experimentación numérica comparando los resultados de las preguntas 2 y 3.
- Avance de la monografía

04/07 y 06/07

- Preguntas 4, 5 y 6.
- Experimentación numérica comparando los resultados de las preguntas 2, 3 y 4; y de las preguntas 5 y 6.
- Versión final de la monografía

7. Puntajes

1. Monografía 10 %
2. Primera entrega 35 %
2. Segunda entrega 55 %

No habrá entregas fuera de las fechas, ni reconsideraciones.

8. Código de honor

El plagio es una falta grave. Cada grupo debe trabajar individualmente. Cualquier código similar entre grupos hará que ambos tengan nota **cero**. De la misma manera, será penado con **cero** copiar código obtenido de otra fuente. Más aún, los alumnos involucrados en un plagio recibirán un descuento de 100 % en su nota final. Además se le informará a las autoridades pertinentes de la universidad.

Referencias

[1] Hopcroft, John E. and Motwani, Rajeev and Ullman, Jeffrey D. Introduction to Automata Theory, Languages, and Computation, 2Nd Edition. *Addison-Wesley Publishing Company.*, 2001.