Universidad de Chile
Departamento de Ciencias de la Computación
CC4102 - Diseño y Análisis de Algoritmos

Tarea 2 - Búsqueda en texto, arreglo de sufijos y autómata

Profesor: Pablo Barceló
Auxiliar: Manuel Cáceres
Ayudantes: Jaime Salas
Claudio Torres

1. Introducción

El objetivo de esta tarea es implementar, evaluar y comparar en la práctica diferentes enfoques para la búsqueda en texto:

- Un enfoque estático basado en **arreglo de sufijos**
- Un enfoque dinámico basado en el **algoritmo con autómata**

Se espera que se implementen los algoritmos y se entregue un informe que indique claramente los siguientes puntos:

1. Las *hipótesis* escogidas antes de realizar los experimentos.
2. El *diseño experimental*, incluyendo los detalles de la implementación de los algoritmos, la generación de las instancias y las medidas de rendimiento utilizadas.
3. La *presentación de los resultados* en forma de una descripción textual, tablas y/o gráficos.
4. El *análisis e interpretación* de los resultados.

2. Arreglo de sufijos

Si $T = t_1 t_2 \dots t_n$ es un texto de largo n, definimos su i-ésimo sufijo como:

$$T_i = t_i t_{i+1} \dots t_n$$

Un **arreglo de sufijos** de un texto T, SA_T, es un arreglo que cumple:

$$SA_T[i] = k \Leftrightarrow T_k \text{ es el i-ésimo en orden lexicográfico entre los sufijos de } T$$

Ejemplo

Sufijos ordenados	Arreglo de sufijos
\$	7
a\$	6
ana\$	4
anana\$	2
banana\$	1
na\$	5
nana\$	3

Figura 1: Orden lexicográfico y arreglo de sufijos correspondientes para el texto $T = banana\$$

Búsqueda de Patrón

Para buscar si un patrón $P = p_1 \dots p_m$ es *substring* del texto notamos que si esto último es cierto entonces P será parte del inicio de algún sufijo de T y por lo tanto, podemos hacer *búsqueda binaria* sobre SA_T en búsqueda de la aparición de P. Finalmente como la comparación toma $\mathcal{O}(m)$ esta búsqueda toma $\mathcal{O}(m \log n)$.

Número de ocurrencias de un Patrón

Podemos encontrar el número de ocurrencias de un patrón haciendo 2 *búsquedas binarias* en búsqueda de la primera y última ocurrencias de P en $\mathcal{O}(m \log n)$.

Construcción del arreglo de sufijos

La forma más simple de obtener el **arreglo de sufijos** es ordenar todos los sufijos de T con un algoritmo basado en comparaciones, como en este caso las comparaciones tienen costo $\mathcal{O}(n)$ este algoritmo de construcción tiene costo $\mathcal{O}(n^2 \log n)$.
En esta tarea implementaremos el siguiente algoritmo de coste $\mathcal{O}(n)$:

Algoritmo de construcción lineal

1. Rellenar con un caracter especial (lexicográficamente menor a los caracteres del alfabeto) al final de T, repetido tantas veces como sea necesario para que el siguiente procedimiento sea correcto.

2. Formar las secuencias de triplas:

$$T_\alpha = (t_1 t_2 t_3) \dots (t_{3i-2} t_{3i-1} t_{3i}) \dots$$
$$T_\beta = (t_2 t_3 t_4) \dots (t_{3i-1} t_{3i} t_{3i+1}) \dots$$
$$T_\gamma = (t_3 t_4 t_5) \dots (t_{3i} t_{3i+1} t_{3i+2}) \dots$$

3. Asignar a cada tripla un caracter en orden lexicográfico y reemplazar en T_α y T_β estos caracteres.

4. Obtener recursivamente el **arreglo de sufijos** de $T_\alpha T_\beta$, $SA_{T_\alpha T_\beta}$, que es un problema de tamaño $\sim \frac{2}{3}n$.

5. Obtener el **arreglo de sufijos** de T_γ, SA_{T_γ}. Para esto podemos hacer `Radix Sort`, considerando que:

$$\begin{aligned}(T_\gamma)_i &= t_{3i} \dots \\ &= t_{3i} t_{3i+1} \dots \\ &= t_{3i} (T_\alpha)_{i+1}\end{aligned}$$

   Es decir, ordenamos palabras de "dos" caracteres, uno dado por un caracter real (t_{3i}) y el resto dado por que conocemos el orden de los sufijos de T_α (obtenido en el punto anterior).

6. Mezclamos $SA_{T_\alpha T_\beta}$ y SA_{T_γ}, con lo que obtenemos y retornamos SA_T. Para lograr hacer esta mezcla en tiempo lineal, necesitamos saber como comparar sufijos de $T_\alpha T_\beta$ con sufijos de T_γ en tiempo constante, esto lo logramos considerando que:

$$(T_\alpha)_i \text{ vs } (T_\gamma)_j \Leftrightarrow t_{3i-2} (T_\beta)_i \text{ vs } t_{3j} (T_\alpha)_{j+1}$$

   que se reduce a una comparación de caracteres y una de sufijos de T_β y T_α que sabemos como resolver gracias a $SA_{T_\alpha T_\beta}$. Y también:

$$(T_\beta)_i \text{ vs } (T_\gamma)_j \Leftrightarrow t_{3i-1} t_{3i} (T_\alpha)_{i+1} \text{ vs } t_{3j} t_{3j+1} (T_\beta)_{j+1}$$

   que se reduce a dos comparaciones de caracteres y una de sufijos de T_α y T_β que sabemos como resolver gracias a $SA_{T_\alpha T_\beta}$.

Finalmente, como el algoritmo mostrado anteriormente toma tiempo lineal y genera una llamada recursiva su costo queda representado por la ecuación de recurrencia:

$$T(n) = \mathcal{O}(n) + T\left(\frac{2}{3}n\right) \Rightarrow T(n) \in \mathcal{O}(n)$$

Y por lo tanto, este algoritmo de construcción tiene costo $\mathcal{O}(n)$.

3. Algoritmo con autómata

En cátedras se vio un algoritmo para búsqueda en texto basado en el uso de un autómata que solo depende del patrón P buscado.

Este autómata queda bien definido por :

- Conjunto de estados $Q = \{0, 1, \ldots, m\}$ siendo 0 el estado inicial y m el final.
- Función de transición

$$\begin{aligned} \delta : \Sigma \times Q \to & \quad Q \\ (q, a) \mapsto & \quad \delta(q, a) \quad = \sigma(P[1:q]a) \\ & \qquad\qquad = \text{largo del sufijo más largo de } P[1:q]a \text{ que es prefijo de } P \end{aligned}$$

Ejemplo

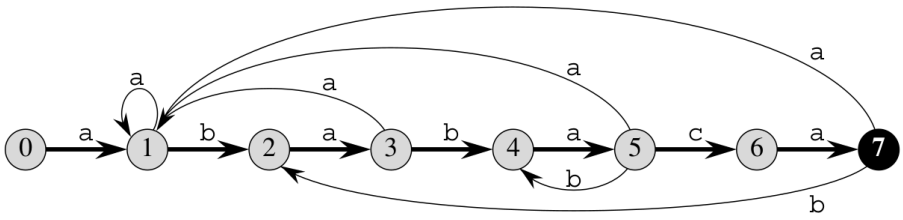

Figura 1: Autómata correspondiente al patrón $P = ababaca$.

Si se ejecuta este autómata sobre el texto T se tienen las siguientes proposiciones:

- Si se alcanza el estado final m, entonces P es substring de T.
- Si se extiende el funcionamiento del autómata para seguir funcionando luego de alcanzar el estado final, entonces el número de veces que aparece P en T es igual al número de veces que se alcanza el estado final en la ejecución del autómata.

Construcción del autómata

Construir el autómata se reduce a construir su función de transición. El algoritmo más simple consiste en hacerlo para cada par $(q, a) \in Q \times \Sigma$ y para cada uno de ellos revisar si $P[1:k]$ es sufijo de $P[1:q]a$, con k desde mín$(q+1, m)$ hasta 0 incurriendo en un costo total de $\mathcal{O}(|\Sigma|m^3) = \mathcal{O}(m^3)$.[1] En esta tarea implementaremos el siguiente algoritmo de coste $\mathcal{O}(m)$:

[1] Asumiendo que el tamaño del alfabeto es una constante. Este algoritmo se encuentra detallado en la segunda edición del libro "Introduction to Algorithms" (CLRS), pág. 922.

Algoritmo de construcción optimizado

La optimización que consideraremos se basa en la siguiente observación:

Al calcular $\delta(q, a) = \sigma(P[1 : q]a)$

- Si $P[q + 1] = a$, entonces $\delta(q, a) = q + 1$.
- Si no ($P[q + 1] \neq a$), entonces $\delta(q, a) = \sigma(P[1 : q]a) = \sigma(P[2 : q]a) \leq q$, y por lo tanto, puede ser calculado corriendo el autómata parcialmente construído sobre $P[2 : q]a$.

Considerando esto podemos construir el autómata en $\mathcal{O}(|\Sigma|m^2) = \mathcal{O}(m^2)$.

Una última observación

En el caso $P[q+1] \neq a$ no es necesario correr el autómata sobre $P[2 : q]a$. Esto se puede lograr manteniendo una variable X que sea el resultado de correr el autómata sobre $P[2 : q]$, es decir, $X = \delta(\ldots\delta(\delta(0, P[2]), P[3])\ldots, P[q])$. Con esto, $\delta(q, a) = \delta(X, a)$, cuando $P[q+1] \neq a$. Obteniendo un tiempo de $\mathcal{O}(|\Sigma|m) = \mathcal{O}(m)$.

4. Pruebas y Datos

Utilice, al menos, textos de lenguaje natural. Preprocese los textos que usará: elimine saltos de línea, puntuación, lleve todo a minúsculas y otras operaciones que estime convenientes. Luego de este preprocesamiento, asegúrese de que sus textos tengan largos (al menos aproximados) de $N = 2^i$ palabras, con $i \in \{15, 16, \ldots, 25\}$.
Para cada texto, construya el **arreglo de sufijos**, midiendo su tiempo de construcción.
Escoja $N/10$ palabras de forma aleatoria del texto y encuentre todas las ocurrencias de éstas, registrando los tiempos de búsqueda en función del largo m del patrón. Recuerde que en el caso del **algoritmo con autómata** cada búsqueda preprocesa el patrón a buscar, tome estos tiempos en función del largo m del patrón. Finalmente, tome los tiempos de `construcción + búsqueda` en cada uno de los métodos y compárelos. Repita estos procesos (construcción y búsqueda) para obtener promedios confiables para los tiempos registrados.
Note que tiene libertad para escoger los textos que utilizará: de esta forma, puede escogerlos para ayudarse a poner a prueba su hipótesis, si fuera necesario. Otro grado de libertad es el tamaño del alfabeto con el que trabajará (por ejemplo, puede comparar lo que sucede al utilizar lenguaje natural versus binario).

5. Entrega de la Tarea

- La tarea puede realizarse en grupos de a lo más 2 personas.
- No se permiten atrasos.
- Para la implementación puede utilizar C, C++ o Java. Para el informe se recomienda utilizar LaTeX.
- Siga buenas prácticas (good coding practices) en sus implementaciones.
- Escriba un informe claro y conciso. Las ponderaciones del informe y la implementación en su nota final son las mismas.
- Tenga en cuenta las sugerencias realizadas en la primera clase auxiliar y en los archivos subidos a U-Cursos sobre la forma de realizar y presentar experimentos.
- La entrega será a través de U-Cursos y deberá incluir el informe junto con el código fuente de la implementación (y todas las indicaciones necesarias para su ejecución).

6. Links

- En http://www.gutenberg.org puede encontrar una gran cantidad de documentos.
- Otra fuente de datos: http://pizzachili.dcc.uchile.cl/texts/nlang/ de lenguaje natural.
- Paper original de construcción de Suffix Array en tiempo lineal: http://web.cs.iastate.edu/~cs548/references/arkk03simple.pdf
- Presentación donde se explica Suffix Array y construcción en tiempo lineal: http://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/suffixarrays.pdf
- Presentación donde se explica el algoritmo con autómata optimizado con un ejemplo: http://www.cs.princeton.edu/courses/archive/spring11/cos226/demo/53KnuthMorrisPratt.pdf [2]

[2] No confundir el algoritmo de Knuth-Morris-Pratt , presentado en: https://www.cs.jhu.edu/~misha/ReadingSeminar/Papers/Knuth77.pdf , con el **algoritmo con autómata** optimizado pedido en esta tarea, el primero toma tiempo $\mathcal{O}(m)$ independiente del alfabeto Σ.