CC3102 Teoría de la Computación
Profesor: Gonzalo Navarro
Auxiliar: Raimundo Lorca Correa
Ayudantes: Nicolás Canales V, Leonel Espinoza, y Belfor Salazar

Tarea 1

11 de abril de 2025

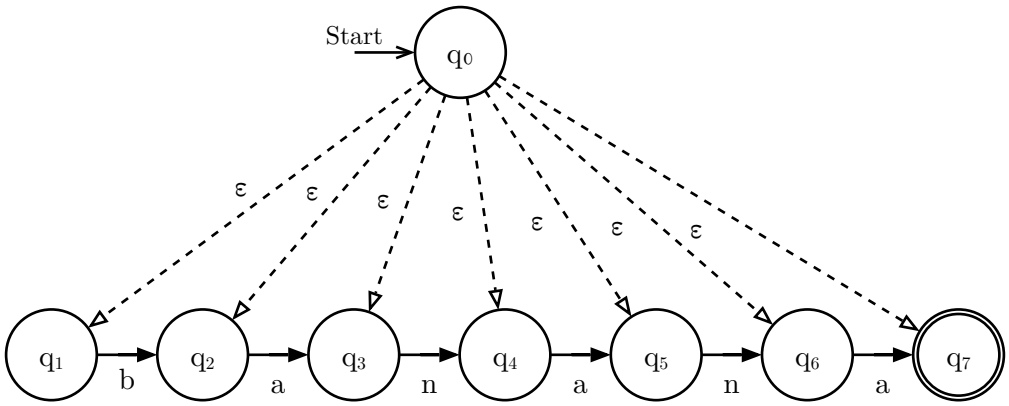
Fecha de entrega: 02 de mayo de 2025.

El propósito principal de esta tarea es explorar una de las aplicaciones más inmediatas de la Teoría de Autómatas: la búsqueda en texto.

Para esto, indagaremos sobre el algoritmo Backwards DAWG Matching (BDM, sección 2.2 de este paper del profesor Navarro), el que utiliza un autómata conocido como Autómata de Sufijos (o Deterministic Acyclic Word Graph) para optimizar su búsqueda y conseguir un tiempo promedio óptimo de $O(n \log(m)/m)$. Además, lo compararemos con otros algoritmos mejor conocidos para búsqueda en texto.

1. Autómata de Sufijos

Un autómata de sufijos de la palabra p, $S(p)$, es un AFD o AFND que reconoce los sufijos de la palabra p. Por ejemplo, para la palabra $p = banana$, un posible $S(p)$ sería como sigue.



Este AFND se puede construir "de atrás para adelante" mediante el siguiente algoritmo:

Algoritmo $S(p = p_1 ... p_m)$:
1. Construir el estado final q_{m+1}.
2. **Para** $i \in \{m, ..., 1\}$:
3. Construir el estado q_i
4. Agregar la transición $\delta(q_i, p_i) = q_{i+1}$
5. Construir el estado inicial q_0
6. **Para** $i \in \{1, ..., m\}$:
7. Agregar la transición $\delta(q_0, \varepsilon) = q_i$

Posteriormente, usted podrá transformar el AFND a un autómata determinista utilizando el

algoritmo visto en clases. Al resultado de este proceso lo llamaremos[1] $DAWG(p)$

2. Algoritmo BDM

El algoritmo BDM utiliza un autómata de sufijos del patrón reverso, $S(p^r)$, para encontrar cuándo puede saltar caracteres en vez de revisarlos.

El algoritmo se ve como sigue:

Algoritmo $BDM(p = p_1...p_m, T = t_1...t_n)$:

Preprocesamiento:
  Construir $D \leftarrow DAWG(p^r = p_m...p_1)$
Búsqueda:
  $pos \leftarrow 0$
  Mientras $pos \leq n - m$ **hacer**:
    $j \leftarrow m$, $last \leftarrow m$
    $estado \leftarrow D.q_0$
    Mientras $estado \neq \perp$ **hacer**:
      $estado \leftarrow D.\delta(estado, t_{pos+j})$
      $j \leftarrow j - 1$
      Si $estado \in D.F$ **entonces**: // Revisar en tiempo $O(1)$
        Si $j > 0$ **entonces** $last \leftarrow j$
        Sino reportar ocurrencia de p **en la posición** $pos + 1$
      Fin del Si
    Fin del Mientras
    $pos \leftarrow pos + last$
  Fin del Mientras

La idea es que el algoritmo funciona con "ventanas" del texto que cubren $t_{pos+1}...t_{pos+m}$. Cuando se descubre que la ventana leída no está contenida en el patrón, se salta a la última posición en que la ventana leída fue un prefijo del patrón.

2.1. Comparaciones: Knuth-Morris-Pratt y Boyer-Moore

Para validar la eficiencia del algoritmo BDM y tener otro punto de referencia, indague sobre los algoritmos de Knuth-Morris-Pratt (KMP, Wikipedia) y Boyer-Moore (BM, Wikipedia). No es necesario que los implemente, puede conseguir una implementación de internet.

3. Objetivos

Para esta tarea, deberá implementar:

- Un método para construir el DAWG para un patrón arbitrario
- El algoritmo BDM utilizando el DAWG creado anteriormente

[1] Abusando de la notación, pues realmente este nombre describe una versión comprimida del autómata, lo que escapa el alcance de esta tarea.

- Los algoritmos KMP y BM. Puede obtener implementaciones de internet, pero deben estar en el mismo lenguaje para que la comparación sea justa.
- Evaluar los costos de búsqueda con los distintos algoritmos.

Con esto, podremos evaluar la eficiencia del algoritmo BDM y compararla con otros algoritmos

4. Experimentación

Experimentaremos inspirándonos en cadenas de ADN, utilizando textos sobre el alfabeto $\Sigma = \{A, C, T, G\}$. Para esto, cree textos aleatorios de largo $n = 2^{20}$ y patrones a buscar de largo $m = 2^j$ con $j \in \{6, 7, 8, 9, 10\}$. Recomendamos crear un texto inicial T y extraer patrones aleatoriamente desde éste, haciendo $p = T[i]...T[i + 2^j - 1]$ para algún i válido (es decir, que no tope con el final del texto).

Para cada j, ejecute los tres algoritmos y evalúe sus desempeños. Repita este proceso 10 veces (con textos y patrones nuevos cada vez) para obtener promedios representativos. Grafique los tiempos obtenidos con los tres algoritmos en función del valor de j.

5. Entregables

Se deberá entregar el código y un informe donde se explique el experimento en estudio. Con esto obtendrá una nota de código (N_{Cod}) y una nota de informe (N_{Inf}). La nota de la tarea será:

$$NT_1 = 0.5N_{Cod} + 0.5N_{Inf}$$

5.1. Código

La entrega de código será en C, C++, Java, o Python. Debe contener:

- (**0.5 pts**) README: Archivo con las instrucciones para ejecutar el código. Debe ser lo suficientemente explicativo para que una persona razonable pueda ejecutar la totalidad de su código sólo leyendo el README.
- (**0.5 pts**) Experimento: Creación de textos y patrones a utilizar en los experimentos, basándose en la cantidad pedida en este enunciado.
- (**1.0 pts**) Construcción del Autómata de Sufijos
- (**1.5 pts**) Transformación a un Autómata Finito Determinista
- (**2.0 pts**) Implementación del algoritmo BDM
- (**0.5 pts**) Main: Un archivo o parte del código (función main) que permita ejecutar los distintos algoritmos y experimentos.

5.2. Informe

- (**0.8 pts**) Introducción: Presentación del tema en estudio, resumir lo que dirá el informe y presentar una hipótesis.
- (**0.8 pts**) Desarrollo: Presentación de algoritmos, estructuras de datos, en lo que se diferencian los algoritmos/estructuras y cómo funcionan y por qué. Recordar que los métodos ya son conocidos por el equipo docente, lo que importa son sus propias implementaciones.
- (**2.4 pts**) Resultados: Especificación de los datos que se utilizaron para los experimentos, la cantidad de veces que se realizaron los tests, con qué inputs, que tamaño, etc. Se deben mostrar gráficos/tablas y mencionar solo lo que se puede observar de estos. Se deben mostrar los valores y parámetros que se están usando.
- (**1.2 pts**) Análisis: Comentar y concluir sus resultados. Se hacen las inferencias de sus resultados.

- (**0.8 pts**) Conclusión: Recapitulación de lo que se hizo, se concluye lo que se puede decir con respecto a sus resultados. También ven si su hipótesis se cumplió o no y analizan la razón. Por último, se menciona qué se podría mejorar en su desarrollo en una versión futura, qué falta en su documento, qué no se ha resuelto y cómo se podría extender.