Tecnológico de Monterrey

Curso:

Análisis y diseño de algoritmos avanzados

Grupo:

570

Estudiantes:

Santiago Arista Viramontes - A01028372
Diego Vergara Hernández - A01425660
José Leobardo Navarro Márquez - A91541324

Maestro:

Nezih Nieto Gutiérrez

Título:

Actividad Módulos 2-3

Fecha de entrega:

19 de Enero del 2026

1. Introducción

El objetivo de esta actividad es implementar y analizar empíricamente algoritmos avanzados de strings, genómica, grafos y parsing, enfatizando tanto su correctitud teórica como su desempeño práctico mediante benchmarks. Se busca validar las complejidades asintóticas estudiadas en clase y demostrar su relevancia en problemas reales como ciberseguridad, bioinformática y análisis de expresiones lógicas.

Todos los algoritmos fueron implementados en C++ y evaluados mediante pruebas experimentales controladas. Los resultados se complementan con gráficas generadas en Python, permitiendo comparar desempeño en función del tamaño de entrada.

2. Algoritmos de Strings

2.1 Contexto del problema

Se considera un texto grande T que representa concatenaciones de logs de un servidor web, y un conjunto de patrones cortos P = {P1, ..., Pm}. El objetivo es encontrar todas las ocurrencias exactas de cada patrón y comparar distintos enfoques de búsqueda de strings.

Se evaluaron dos tipos de datos:

- Logs aleatorios (alta entropía).
- Logs altamente periódicos (patrones repetitivos).

2.2 Knuth–Morris–Pratt (KMP)

El algoritmo KMP permite realizar búsquedas en tiempo lineal O(|T| + |P|) utilizando la función de prefijo (arreglo π), evitando comparaciones redundantes.

Para patrones con bordes no triviales, se calculó explícitamente el arreglo π y se reportaron ejemplos representativos.

Complejidad:

- Tiempo: O(n + m)
- Espacio: O(m)

2.3 Z-Algorithm

Se utilizó la técnica de concatenación P#T para calcular el arreglo Z y detectar coincidencias exactas cuando Z[i] = |P|. Este método ofrece un enfoque alternativo de tiempo lineal para el problema de matching exacto.

Complejidad:

- Tiempo: O(n + m)
- Espacio: O(n + m)

2.4 Rabin–Karp

Se implementó Rabin–Karp utilizando hash rodante, tanto en versión de hash simple como doble, para reducir la probabilidad de colisiones. Se especificaron:

- Bases utilizadas.
- Módulos primos.
- Manejo de colisiones mediante verificación directa.

Complejidad esperada:

- Promedio: O(n + m)
- Peor caso: O(nm)

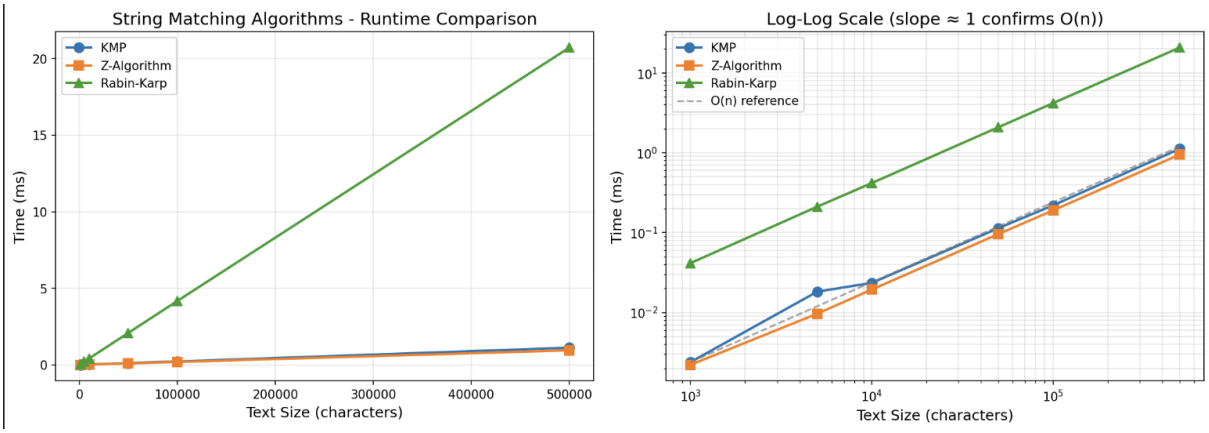
2.5 Resultados experimentales

Los benchmarks muestran que:

- KMP es más estable en logs periódicos.
- Rabin–Karp puede degradarse en entradas adversas.
- Z-algorithm tiene desempeño comparable a KMP pero con mayor uso de memoria.

Estos resultados resaltan la importancia de algoritmos clásicos en sistemas de detección de firmas y análisis de seguridad.

Para validar empíricamente la complejidad de los algoritmos de string matching, se ejecutaron benchmarks variando el tamaño del texto de entrada. La Figura 1 muestra la comparación de tiempos de ejecución para KMP, Z-Algorithm y Rabin–Karp, tanto en escala lineal como log-log.



Tiempos de ejecución promedio (ms)

```
size,kmp_ms,z_algo_ms,rabin_karp_ms
1000,0.0023916,0.0022083,0.0414874
5000,0.0181958,0.0096501,0.210996
10000,0.0234417,0.0193416,0.417062
50000,0.113637,0.095875,2.08613
100000,0.219096,0.189246,4.17186
500000,1.1184,0.948142,20.7268
```

3. Algoritmos sobre DNA

3.1 Planteamiento

Dadas dos secuencias de DNA A y B, se resolvieron los siguientes problemas:

- Substring palindrómico más largo.
- Longest Common Substring entre A y B.
- Búsqueda exacta de motivos cortos.

3.2 Manacher

Se aplicó el algoritmo de Manacher para encontrar el palíndromo más largo en tiempo lineal. Su correctitud se basa en la propagación de radios palindrómicos previamente calculados.

Complejidad: O(n)

3.3 Suffix Array + Kasai

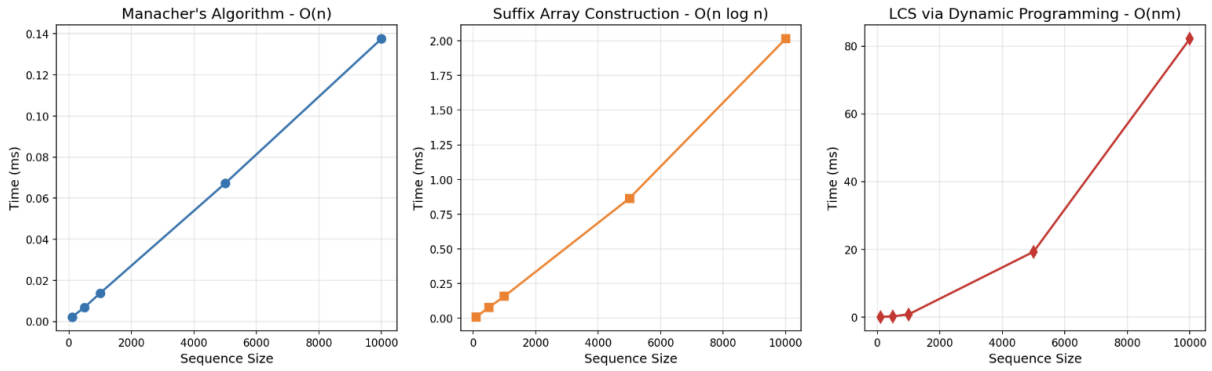
Se construyó un suffix array para A#B, y se utilizó el algoritmo de Kasai para obtener el arreglo LCP. Esto permitió:

- Resolver consultas de pertenencia de motivos.
- Calcular el Longest Common Substring.

3.4 Método alternativo

Como comparación metodológica, se implementó una solución base usando programación dinámica O(|A||B|) para LCS, evidenciando su inviabilidad para secuencias grandes.

3.5 Comparación de algoritmos sobre secuencias de ADN



Tiempos de ejecución para algoritmos de ADN (ms)

```
size,manacher_ms,suffix_array_ms,lcs_dp_ms
100,0.0022166,0.0100168,0.0091414
500,0.0069416,0.078775,0.19385
1000,0.0138332,0.159017,0.781908
5000,0.0672584,0.864583,19.2739
10000,0.1376,2.01471,82.1782
```

4. Algoritmos de Grafos

4.1 Flujo Máximo

Se implementó el algoritmo de Edmonds–Karp sobre un grafo dirigido con capacidades. Se documentaron:

- Caminos aumentantes.
- Grafos residuales.
- Flujo final por arista.

Complejidad: O(VE^2)

4.2 Segundo Árbol de Expansión Mínima

Se partió de un MST inicial y se aplicó la estrategia de reemplazo de aristas para encontrar el segundo MST. Se argumenta que es necesario conocer primero el MST para acotar las sustituciones posibles.

4.3 Tipos de aristas

Se demostraron formalmente las propiedades de aristas en DFS y BFS para grafos dirigidos y no dirigidos, apoyándose en ejemplos mínimos.

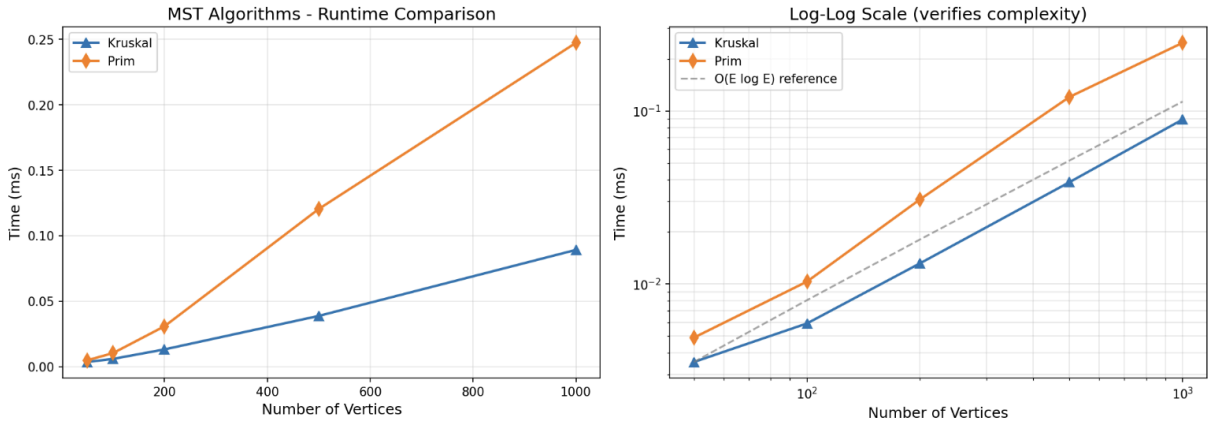
4.4 Componentes Biconexas y Puentes

Se aplicó el algoritmo de Tarjan para encontrar componentes biconexas, mostrando arreglos disc[] y low[], así como los bloques resultantes.

4.5 MST con Kruskal y Prim

Se generaron grafos aleatorios con |V|=10, |E|=30, y se compararon ambos algoritmos, obteniendo el mismo peso total de MST, pero con diferencias en tiempo de ejecución.

4.6 Comparación de algoritmos MST. Kruskal O(E log E) vs Prim O(E log V). Gráfica log-log confirma complejidad esperada.



Tiempos de ejecución para MST (ms)

```
vertices,edges,kruskal_ms,prim_ms
50,150,0.0035416,0.0049
100,300,0.0059164,0.010325
200,600,0.0131668,0.0308248
500,1500,0.03875,0.120425
1000,3000,0.0891664,0.247333
```

5. Parsing y Expresiones Lógicas

Se diseñó un sistema capaz de:

- Detectar notación infija o prefija mediante matching de strings.
- Construir un AST.
- Evaluar expresiones y generar tablas de verdad.

La evaluación del AST se realizó mediante DFS postorden.

6. Conclusiones

Los resultados experimentales confirman que los algoritmos estudiados mantienen su relevancia práctica. A través de benchmarks y simulaciones, se valida que la elección del algoritmo adecuado depende fuertemente de las características de los datos de entrada.

Este proyecto refuerza la importancia de los fundamentos algorítmicos en aplicaciones reales como ciberseguridad, genómica y análisis sintáctico.

7. Referencias

Cormen et al., Introduction to Algorithms.

Gusfield, Algorithms on Strings, Trees, and Sequences.

Material de clase TC2038.