Información General

Valor:	20 % de la calificación final del curso
Modalidad:	Grupos de 2 o 3 estudiantes
Lenguaje:	C (estándar C11 o C99)

1 Descripción del proyecto

En este proyecto, los estudiantes implementarán y analizarán algoritmos avanzados de procesamiento de texto y búsqueda de patrones, aplicándolos a un problema práctico de análisis de documentos y búsqueda de información. El objetivo es desarrollar un sistema eficiente que permita indexar, buscar y analizar grandes volúmenes de texto, evaluando el rendimiento de los algoritmos implementados.

2 Objetivos de aprendizaje

- Implementar correctamente algoritmos avanzados de búsqueda de patrones en texto
- Desarrollar estructuras de datos eficientes para indexación y recuperación de información
- Analizar empíricamente la complejidad y rendimiento de los algoritmos
- Aplicar los algoritmos a un problema práctico de procesamiento de documentos
- Implementar técnicas de preprocesamiento de texto para mejorar la eficiencia de búsqueda
- Documentar adecuadamente el código y los resultados del análisis

3 Requisitos técnicos

3.1 Algoritmos de búsqueda de patrones

Implementar tres (3) de los siguientes algoritmos de búsqueda de patrones en texto (a elección del grupo):

1. **Algoritmo de Knuth-Morris-Pratt (KMP):**
   - Implementación completa con preprocesamiento de patrones
   - Análisis de la función de fallos
   - Optimizaciones para mejorar el rendimiento
2. **Algoritmo de Boyer-Moore:**
   - Implementación con heurística de carácter malo
   - Implementación con heurística de sufijo bueno
   - Versión completa con ambas heurísticas

3. **Algoritmo Shift-And (Bitap):**
    - Implementación basada en operaciones de bits (AND bit a bit)
    - Optimización para patrones cortos
    - Extensión para búsqueda aproximada con distancias de edición
4. **Algoritmo Shift-Or:**
    - Implementación basada en operaciones de bits (OR bit a bit)
    - Comparación de rendimiento con Shift-And
    - Optimizaciones para diversos casos de uso
5. **Algoritmo de Aho-Corasick:**
    - Construcción del autómata de patrones múltiples
    - Implementación de la función de fallo
    - Búsqueda eficiente de múltiples patrones simultáneamente

3.2 Estructuras de datos para indexación

Implementar al menos dos de las siguientes estructuras de datos:

1. **Trie (Árbol de Prefijos):**
    - Implementación básica para almacenar palabras
    - Operaciones de inserción, búsqueda y eliminación
    - Optimización de memoria (trie compacto o radix tree)
2. **Índice Invertido:**
    - Construcción del índice a partir de documentos
    - Operaciones de búsqueda y actualización
    - Soporte para consultas booleanas
3. **Tabla Hash para Texto:**
    - Implementación de funciones hash adecuadas para texto
    - Manejo de colisiones
    - Optimización para grandes volúmenes de texto
4. **Filtro de Bloom:**
    - Implementación con múltiples funciones hash
    - Configuración de tamaño óptimo según tasa de falsos positivos
    - Aplicación para prefiltraje de búsquedas

3.3 Técnicas de preprocesamiento de texto

Implementar al menos dos (2) de las siguientes técnicas de preprocesamiento:

1. **Tokenización:**
    - División de texto en palabras, frases o n-gramas
    - Manejo de caracteres especiales y puntuación
    - Soporte para diferentes idiomas
2. **Normalización:**
    - Conversión a minúsculas/mayúsculas
    - Eliminación de acentos y caracteres diacríticos
    - Normalización de espacios y caracteres especiales
3. **Eliminación de Stopwords:**
    - Implementación de listas de stopwords configurables
    - Soporte para múltiples idiomas
    - Filtrado por longitud o frecuencia
4. **Stemming o Lematización:**
    - Implementación de un algoritmo de stemming simple
    - Reducción de palabras a su forma base
    - Aplicación para mejorar la recuperación de información

3.4 Framework de análisis de rendimiento

Desarrollar un sistema para medir y comparar el rendimiento de los algoritmos:

- Medición precisa de tiempos de ejecución
- Conteo de operaciones clave (comparaciones, accesos a memoria)
- Generación de conjuntos de datos de prueba con diferentes características
- Visualización de resultados (tablas y gráficos)

4 Aplicación práctica: Sistema avanzado de análisis de documentos y procesamiento lingüístico para línea de comandos

Implementar un sistema completo de análisis de documentos que funcione exclusivamente desde la línea de comandos, con las siguientes funcionalidades:

1. **Carga y preprocesamiento de documentos:**
    - Soporte para archivos de texto plano y HTML simple
    - Extracción de texto y metadatos
    - Aplicación de técnicas de preprocesamiento

2. **Indexación de documentos:**
    - Construcción de índices para búsqueda rápida
    - Actualización incremental de índices
    - Persistencia de índices en disco
3. **Motor de búsqueda:**
    - Búsqueda exacta de palabras y frases (obligatorio)
    - Búsqueda aproximada (con tolerancia a errores) (recomendado). Si la implementa, puede utilizar algoritmos como la distancia de Levenshtein o técnicas de n-gramas
    - Consultas booleanas y de proximidad
    - Ranking de resultados por relevancia
4. **Análisis de texto:**
    - Estadísticas de documentos (frecuencia de palabras, longitud, etc.)
    - Detección de palabras clave
    - Análisis de similitud entre documentos
5. **Interfaz de línea de comandos:**
    - Programa ejecutable desde terminal con argumentos y opciones bien definidos (sin menú interactivo)
    - Sistema de comandos completo que permita acceder a todas las funcionalidades directamente
    - Parámetros y flags consistentes con las convenciones estándar de Unix
    - Visualización de resultados con formato adecuado para terminal (tablas ASCII, colores)
    - Generación de archivos CSV o JSON para análisis posterior de los resultados
    - Soporte para redirección de salida a archivos y tuberías (pipes)
    - Documentación de ayuda accesible mediante flag –help o -h

5 ENTREGABLES

1. **Código fuente:**
    - Archivos .c y .h bien organizados y comentados
    - Estructura modular con separación clara de responsabilidades
2. **Repositorio GitHub:**
    - Repositorio público con historial de commits de todos los integrantes
    - Makefile estándar para compilación del proyecto
    - Archivo README.md con instrucciones detalladas del proyecto
    - Especificar las responsabilidades de cada integrante
3. **Datos de prueba:**

- Conjunto diverso de textos y patrones para probar el sistema, incluyendo casos especiales
- Scripts para generar datos sintéticos y casos de prueba
- Documentación de los casos de prueba y su propósito

4. **Informe técnico:** Documento PDF (8-20 páginas) que incluya:
   - Análisis teórico de la complejidad de los algoritmos
   - Resultados experimentales con gráficos comparativos
   - Discusión de los resultados y conclusiones
   - Comparativa de rendimiento entre los algoritmos implementados
5. **Documentación de código:** Comentarios claros en el código fuente que expliquen el funcionamiento de las funciones y argumentos del programa

6 Presentación oral

Como parte integral de la evaluación, cada grupo deberá realizar una presentación oral de su proyecto, con las siguientes características:

- Duración: 10-15 minutos máximo
- Participación: Todos los integrantes del grupo deben participar activamente
- Contenido: Diseño del sistema, algoritmos implementados, estructuras de datos, resultados experimentales
- Preguntas: Capacidad de responder a consultas técnicas del profesor y compañeros

7 Sistema de evaluación

El proyecto será evaluado considerando dos componentes principales: el código (60 %) y el informe técnico (40 %). La nota del informe técnico estará afectada por un factor multiplicador dependiente de la presentación oral.

7.1 Rúbrica para evaluación del código (60 %)

7.2 Rúbrica para evaluación del informe técnico (40 %)

7.3 Factor multiplicador de la presentación oral

La nota del informe técnico será modificada por un factor multiplicador basado en la calidad de la presentación oral:

$$\textbf{Nota Final del Informe} = \textbf{Nota Original del Informe} \times \textbf{Factor de Presentación}$$

7.4 Valoración de originalidad y material extra

Se premiarán especialmente las ideas originales y el material o código extra que enriquezca el proyecto más allá de los requisitos mínimos:

- **Ideas originales:** Implementaciones novedosas de los algoritmos, optimizaciones creativas o enfoques alternativos que demuestren un pensamiento innovador

Criterio	Excelente (90-100 %)	Bueno (75-89 %)	Satisfactorio (60-74 %)	Insuficiente (0-59 %)
Correctitud (35 %)	Implementación correcta de todos los algoritmos y estructuras con manejo de casos borde.	Implementación correcta con buen manejo de casos comunes.	Implementación funcional con errores menores.	Implementación con errores significativos.
Eficiencia (25 %)	Código altamente optimizado con excelente rendimiento computacional.	Buen rendimiento y uso eficiente de recursos.	Rendimiento aceptable para casos comunes.	Código ineficiente con problemas de rendimiento.
Calidad (25 %)	Estructura modular excelente, alta legibilidad y manejo robusto de errores.	Buena estructura, código legible y manejo adecuado de errores.	Estructura básica funcional con legibilidad aceptable.	Código desorganizado o difícil de mantener.
Funcionalidad (15 %)	Implementación completa de todas las funcionalidades requeridas con interfaz robusta.	Implementación de la mayoría de funcionalidades con buena interfaz.	Implementación básica con interfaz funcional.	Funcionalidades incompletas o interfaz deficiente.

Cuadro 1: Rúbrica para la evaluación del código

- **Código extra:** Funcionalidades adicionales, interfaces más elaboradas, visualizaciones, o implementación de algoritmos opcionales
- **Material adicional:** Estudios comparativos más extensos, análisis de casos de uso reales, o demostraciones prácticas de aplicaciones

7.5 Evaluación de pares

Como parte del proceso de evaluación, se implementará un sistema de evaluación de pares donde cada integrante del grupo evaluará la contribución y desempeño de sus compañeros de equipo:

- Cada integrante evaluará individualmente a los demás miembros de su propio grupo
- La evaluación se realizará con una escala de 0 a 7, donde 7 representa un desempeño sobresaliente
- La evaluación incluirá aspectos como contribución al código, participación en reuniones, cumplimiento de responsabilidades asignadas y calidad del trabajo
- Las evaluaciones serán confidenciales y entregadas directamente al profesor el día de la presentación oral

Criterio	Excelente (90-100 %)	Bueno (75-89 %)	Satisfactorio (60-74 %)	Insuficiente (0-59 %)
Originalidad (30 %)	Texto completamente original con citas adecuadas.	Texto mayormente original con referencias correctas.	Texto original con algunas citas imprecisas.	Plagio parcial o total (descalificación).
Análisis teórico (30 %)	Fundamentación matemática rigurosa y completa.	Buen análisis teórico con base matemática sólida.	Análisis teórico básico pero correcto.	Análisis superficial o con errores conceptuales.
Experimentación (30 %)	Metodología rigurosa con análisis estadístico detallado.	Buena metodología con análisis adecuado de resultados.	Experimentos básicos con análisis simple.	Experimentos insuficientes o mal analizados.
Presentación (10 %)	Estructura excelente, redacción clara y gráficos de alta calidad.	Buena estructura, redacción y recursos visuales adecuados.	Estructura coherente y redacción aceptable.	Mala organización o redacción confusa.

Cuadro 2: Rúbrica para la evaluación del informe técnico

- Esta evaluación constituirá un 10 % de la nota final individual de cada estudiante

Este proceso promueve la responsabilidad individual, el trabajo equitativo y la transparencia en las contribuciones de cada miembro al proyecto grupal.

7.6 Cálculo de la nota final

Nota Final = [(Nota del Código $\times$ 0.6) + (Nota del Informe $\times$ Factor de Presentación $\times$ 0.4)] $\times$ 0.9 + (Evaluación de Pares $\times$ 0.1)

8 Consejos y recursos

- Comience implementando versiones básicas de los algoritmos y luego añada optimizaciones.
- Para probar los algoritmos de búsqueda, utilice textos de diferentes tamaños y patrones con diferentes características.
- Preste especial atención a la gestión de memoria, especialmente para las estructuras de indexación con grandes volúmenes de texto.
- Para la visualización de resultados, puede utilizar herramientas como Gnuplot o exportar datos para procesarlos con Python (matplotlib).
- Recursos recomendados:
  - Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press. (Capítulos sobre algoritmos de texto)
  - Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.
  - Crochemore, M., Hancart, C., & Lecroq, T. (2007). Algorithms on Strings. Cambridge University Press. (Referencia fundamental para algoritmos de coincidencia exacta)

Situación	Factor	Descripción
No presentación	0.0	La no realización de la presentación implica un factor 0, anulando completamente la nota del informe técnico.
Presentación deficiente	0.5	Presentación desorganizada, explicaciones confusas, incapacidad para responder preguntas básicas.
Presentación regular	0.7	Presentación estructurada pero con explicaciones superficiales y respuestas parciales a las preguntas.
Presentación adecuada	1.0	Presentación clara con buena estructura, explicaciones adecuadas y respuestas satisfactorias.
Presentación excelente	1.2	Presentación sobresaliente con dominio excepcional del tema.

Cuadro 3: Factores multiplicadores para la evaluación del informe técnico

- Lecroq, T. (2007). Fast exact string matching algorithms. Information Processing Letters, 102(6), 229-235.
- Gusfield, D. (1997). Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press.
- Repositorio de algoritmos de procesamiento de texto: https://github.com/OpenGenus/cosmos/tree/master/code/string_algorithms
- **Corpus de texto para pruebas:**
  - Project Gutenberg: https://www.gutenberg.org/ (textos literarios libres de derechos)
  - Common Crawl: https://commoncrawl.org/ (gran corpus de páginas web)
  - TREC Collections: https://trec.nist.gov/data.html (colecciones estándar para IR)
  - Wikimedia Dumps: https://dumps.wikimedia.org/ (datos de Wikipedia y proyectos relacionados)
  - Corpus del Español: https://www.corpusdelespanol.org/ (especial para textos en español)
  - Leipzig Corpora Collection: https://wortschatz.uni-leipzig.de/en/download/ (múltiples idiomas)
  - SMS Spam Collection: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset (para probar con textos cortos)
- Biblioteca de algoritmos de strings: https://github.com/s-yata/darts-clone (para implementaciones de referencia)
- Lecroq, T.: Handbook of Exact String Matching Algorithms: https://www-igm.univ-mlv.fr/~lecroq/string.pdf (Manual completo de algoritmos de coincidencia exacta con pseudocódigo detallado)

9 PREGUNTAS FRECUENTES

1. **¿Podemos utilizar bibliotecas externas?**
   Se permite el uso de bibliotecas estándar de C. Para bibliotecas adicionales, consulte con el instructor. Sin embargo, los algoritmos de búsqueda y las estructuras de datos deben ser implementados por usted.

2. **¿Cómo debemos manejar textos en diferentes idiomas?**
   Para simplificar, puede centrarse en el soporte para texto en español e inglés. Si desea manejar otros idiomas, considere la codificación UTF-8 y las particularidades de cada idioma.

3. **¿Qué tamaño deben tener los documentos para las pruebas?**
   Para un análisis significativo, utilice documentos de diferentes tamaños, desde pequeños (pocos KB) hasta grandes (varios MB). Incluya también colecciones de múltiples documentos.

4. **¿Es necesario implementar búsqueda aproximada?**
   La búsqueda exacta es obligatoria. La búsqueda aproximada (con tolerancia a errores) es recomendada. Si la implementa, puede utilizar algoritmos como la distancia de Levenshtein o técnicas de n-gramas.