UNIVERSIDAD DE GUADALAJARA
CUCEI

Proyecto Final

Materia: Análisis de algoritmos

Actividad: Proyecto Final 25b

Equipo: Lobas

Alumnos:

Cuéllar Hernández Cinthya Sofía
22391799
Hernández Santos Karen Cecilia
219770168
Valentín Gallardo José Eduardo
218452685

Introducción

El análisis de secuencias genómicas es una actividad fundamental dentro de la bioinformática, ya que es posible identificar patrones, estructuras y propiedades importantes dentro del ADN. Uno de los problemas más estudiados es la búsqueda del palíndromo más largo dentro de una secuencia genética, ya que los palíndromos cumplen funciones clave en la regulación de genes, en la formación de configuraciones secundarias y en la delimitación de sitios de restricción utilizados por enzimas.

En este proyecto desarrollamos un sistema que compara diferentes algoritmos que permiten identificar el mayor palíndromo existente en una sub secuencia del genoma. El algoritmo base inicia con Fuerza Bruta, y poco a poco se integran otras técnicas vistas en el curso, como: Divide y Vencerás, Programación Dinámica y Algoritmos Voraces (Greedy). Cada una de estas técnicas se implementan, analizan y comparan en términos de desempeño computacional, efectividad y practicidad.

Las pruebas se realizaron sobre un archivo genómico real en formato **.fna**, obtenido de una base de datos especializada. Esto permitió observar diferencias tangibles entre las metodologías y apreciar cómo varía su desempeño conforme aumenta el tamaño de la entrada.

Objetivo General

En este proyecto, nuestro objetivo es implementar, analizar y comparar distintas técnicas de diseño de algoritmos para resolver el problema de encontrar el palíndromo más largo en una secuencia genómica, evaluando y comparando su eficiencia, precisión y comportamiento computacional.

Objetivos Específicos

1. Implementar el algoritmo base mediante Fuerza Bruta.
2. Integrar la técnica Divide y Vencerás para mejorar el rendimiento.
3. Implementar una solución basada en Programación Dinámica para optimizar los subproblemas.
4. Desarrollar y analizar un algoritmo Voraz (Greedy) basado en expansión desde el centro.
5. Comparar el comportamiento y tiempos de ejecución de todos los métodos.
6. Analizar la escalabilidad de los algoritmos para distintas longitudes de sub secuencia.
7. Documentar los resultados mediante reporte, gráficos y evidencias de ejecución.

Desarrollo

Descripción del problema

Con una secuencia genética obtenida desde un archivo `.fna`, se requiere identificar el palíndromo más largo, es decir, una subcadena que se lee igual de izquierda a derecha que de derecha a izquierda.

Ejemplos clásicos de palíndromos:

- "ACGTGTCA" no es palíndromo
- "ATCGCTA" sí lo es

Este problema es relevante, ya que los palíndromos pueden formar estructuras como bucles, afectando la estabilidad y función del ADN.

Implementación de cada técnica.

Fuerza Bruta

```
def palindromoFuerzaBruta(cadena):
    """...
    if len(cadena) == 0:
        return ""
```

Esta función recorre todas las subcadenas posibles de la cadena original.
Para eso usa dos ciclos anidados:

```
for i in range(len(cadena)):
    for j in range(i + max_longitud, len(cadena) + 1):
        subcadena = cadena[i:j]
        if esPalindromo(subcadena) and len(subcadena) > max_longitud:
            max_longitud = len(subcadena)
            mejor_palindromo = subcadena
```

- `i` marca el inicio de la subcadena.
- `j` marca el final.
- Se construye la subcadena como: `cadena[i:j]`.

Cada subcadena se verifica con la función auxiliar:

esPalindromo(subcadena)

que simplemente compara la cadena consigo misma invertida:

cadena == cadena[::-1].

La función guarda el palíndromo más largo encontrado hasta el momento, se actualiza cuando detecta uno mejor. Además, solo se usa para n ≤ 15, porque su complejidad es muy costosa.

Funciones involucradas

esPalindromo(): revisa si una cadena es palíndroma.

contarRepeticiones(): después de obtener el palíndromo se usa para medir repeticiones dentro de la secuencia.

Complejidad:

- Tiempo: **O(n³)** (generación + verificación)
- Espacio: **O(1)**

Ventajas	**Desventajas**
Encuentra la solución óptima sin fallar.	Inviable para secuencias medianas o grandes.

Divide y Vencerás

1. Dividir la cadena a la mitad.
2. Encontrar el palíndromo más largo en la mitad izquierda.
3. Encontrarlo en la mitad derecha.
4. Expandir desde el centro para detectar palíndromos que cruzan la frontera.
5. Elegir el mayor de los tres.

Complejidad:
Tiempo: **O(n²)**
Espacio: **O(log n)** (por recursión)

Ventajas	**Desventajas**
Reduce el tiempo y mejor rendimiento que Fuerza Bruta. Enfoque recursivo.	Requiere combinar resultados centrales con cuidado. Sin tabla de memorización, puede recalcular regiones.

Programación Dinámica

Idea general:
Usar una tabla `DP[i][j]` que indica si la subcadena desde i hasta j es palíndroma.

Complejidad:

- Tiempo: **O(n²)**
- Espacio: **O(n²)**

Ventajas	**Desventajas**
Muy eficiente a comparación de fuerza bruta. Calcula subproblemas solo una vez. Garantiza resultados óptimos.	Requiere memoria cuadrática.

Algoritmo Voraz (Greedy) - Expansión desde centro

En cada posición de la cadena, tomamos la decisión local de expandirnos lo máximo posible desde ese centro, la elección voraz es: "en este centro, expandirme hasta donde sea palíndromo" y luego seleccionamos el mejor resultado local global**,** los palíndromos solo pueden crecer desde un **centro**, por eso, se hace lo siguiente:

1. Para cada posición `i` se toma como centro (palíndromos impares).
2. También se toma el par `(i, i+1)` como centro para palíndromos pares.
3. Se expande mientras los caracteres coincidan.
4. Se elige el mejor de todas las expansiones locales.

Complejidad

- Tiempo: **O(n²)**
- Espacio: **O(1)**

Ventajas	**Desventajas**
Muy fácil de implementar. Rápido, intuitivo y sin uso extra de memoria. Excelente balance entre eficiencia y claridad	No utiliza técnicas avanzadas como memorización o particiones. No garantiza ser el método más óptimo teórico, pero en práctica funciona muy bien.

Las pruebas se realizaron sobre una secuencia obtenida del archivo genómico real. Adicionalmente, cuando el archivo no estuvo disponible, se utilizó una secuencia sintética repetitiva para poder realizar las pruebas sin afectar la estructura del

experimento.
Se evaluaron sub secuencias de tamaños: 8, 12,16, 20.

Resultados obtenidos:

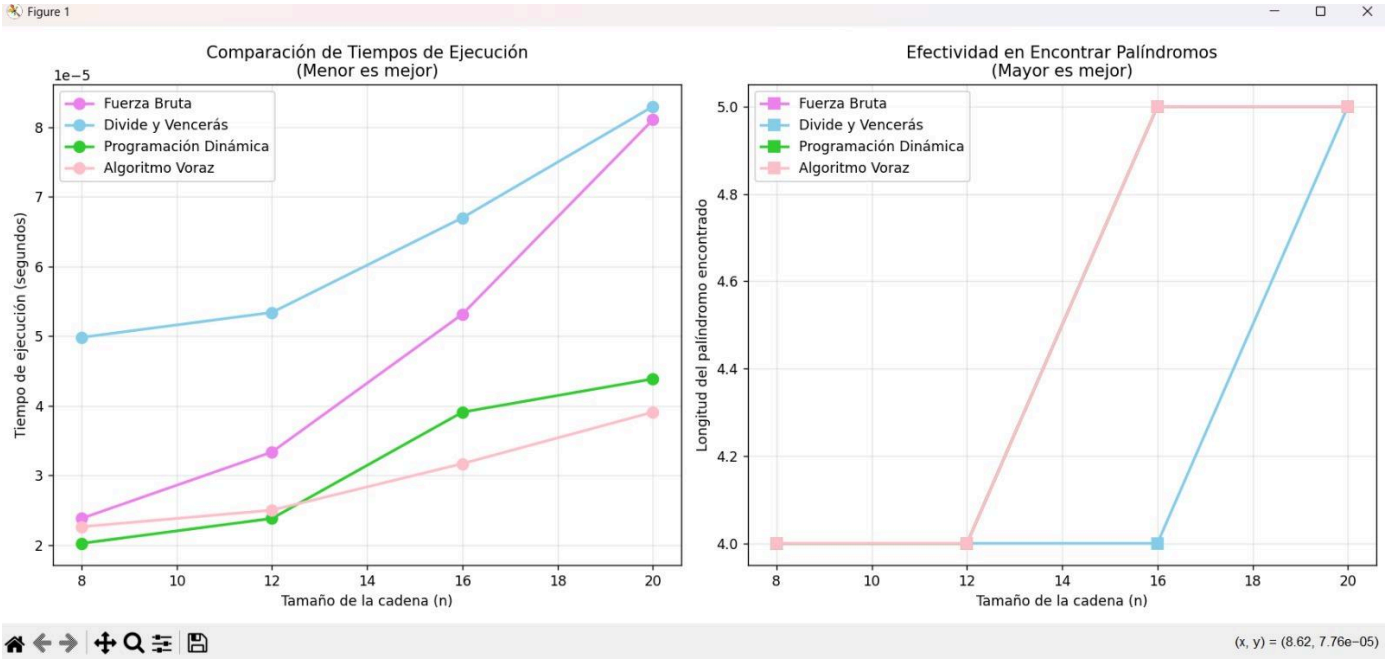
Fuerza Bruta solo pudo ejecutarse hasta n = 15 debido a su complejidad.

Divide y Vencerás mostró una mejora clara respecto a Fuerza Bruta.

Programación Dinámica siempre encontró la misma solución óptima que el resto de los métodos.

Voraz obtuvo los tiempos más bajos en todas las pruebas..

Comparativas:




```
Se leyeron las primeras 20 lineas del archivo (1600 bases)
Primeros 50 caracteres: ATATTAGGTTTTTACCTACCCAGGAAAAGCCAACCAACCTCGATCTCTTG...
Comparación de algoritmmos para la busqueda de palindromos


Longitud n = 8

Fuerza Bruta: 'ATTA' (4 bases)
  Repeticiones: 7, Tiempo: 0.00002s
Divide y Vencerás: 'ATTA' (4 bases)
  Repeticiones: 7, Tiempo: 0.00005s
Programación Dinámica: 'ATTA' (4 bases)
  Repeticiones: 7, Tiempo: 0.00002s
Algoritmo Voraz: 'ATTA' (4 bases)
  Repeticiones: 7, Tiempo: 0.00002s
```

```
Longitud n = 12

Fuerza Bruta: 'ATTA' (4 bases)
  Repeticiones: 7, Tiempo: 0.00003s
Divide y Vencerás: 'ATTA' (4 bases)
  Repeticiones: 7, Tiempo: 0.00005s
Programación Dinámica: 'ATTA' (4 bases)
  Repeticiones: 7, Tiempo: 0.00002s
Algoritmo Voraz: 'ATTA' (4 bases)
  Repeticiones: 7, Tiempo: 0.00003s


Longitud n = 16

Fuerza Bruta: 'TTTTT' (5 bases)
  Repeticiones: 1, Tiempo: 0.00005s
Divide y Vencerás: 'ATTA' (4 bases)
  Repeticiones: 7, Tiempo: 0.00007s
Programación Dinámica: 'TTTTT' (5 bases)
  Repeticiones: 1, Tiempo: 0.00004s
Algoritmo Voraz: 'TTTTT' (5 bases)
  Repeticiones: 1, Tiempo: 0.00003s
```

```
Longitud n = 20

Fuerza Bruta: 'TTTTT' (5 bases)
  Repeticiones: 1, Tiempo: 0.00008s
Divide y Vencerás: 'TTTTT' (5 bases)
  Repeticiones: 1, Tiempo: 0.00008s
Programación Dinámica: 'TTTTT' (5 bases)
  Repeticiones: 1, Tiempo: 0.00004s
Algoritmo Voraz: 'TTTTT' (5 bases)
  Repeticiones: 1, Tiempo: 0.00004s

Process finished with exit code -1
```

El programa genera dos gráficos principales:

1. **Tiempo de ejecución vs tamaño de entrada**
    - Fuerza Bruta crece abruptamente.
    - Divide y Vencerás y PD se comportan de forma cuadrática.
    - Voraz es consistentemente el más rápido.
2. **Longitud del palíndromo encontrado**
    - Todos los métodos coinciden en el mismo resultado.
    - Muestra consistencia entre técnicas.

Conclusiones

El proyecto nos permitió analizar en profundidad distintas técnicas de diseño de algoritmos aplicadas a un problema real de bioinformática. Entre los hallazgos más relevantes pudimos ver cómo cada enfoque se comporta frente a datos reales y qué ventajas o limitaciones aparecen en la práctica.

1. La fuerza bruta asegura siempre la solución más óptima encontrada, pero su costo computacional es alto, lo que hace a este algoritmo poco viable para secuencias medianas a grandes.
2. Divide y Vencerás: este algoritmo logra reducir significativamente el esfuerzo computacional, aunque puede recalcular subproblemas.
3. La Programación Dinámica es una solución poderosa que logra optimizar y evita cómputos repetidos a costa de mayor uso de memoria.
4. El algoritmo Voraz obtuvo el mejor balance general mostrando tiempos menores con una implementación muy simple y clara.
5. Todos los métodos coincidieron en los mismos palíndromos más largos, confirmando la validez del enfoque.

En conclusión, el método Voraz destaca como la mejor alternativa práctica para identificar palíndromos largos en secuencias genómicas debido a su eficiencia, simpleza y estabilidad.

Referencias

GeeksforGeeks. (s. f.). *Longest Palindromic Substring*. Recuperado de https://www.geeksforgeeks.org/dsa/longest-palindromic-substring/

AlgoTree. (s. f.). *Finding the Longest Palindromic Substring — Dynamic Programming Approach*. Recuperadob de https://www.algotree.org/algorithms/dynamic_programming/longest_palindromic_substring/

GeeksforGeeks. (2025, julio). *Longest Palindromic Substring using Dynamic Programming*. Recuperado de https://www.geeksforgeeks.org/dsa/longest-palindromic-substring-using-dynamic-programming-2/