Universidad de Tecnología Metropolitana

Taller 2 de Análisis de Algoritmo

Ignacio Baeza
Departamento de Ingeniería Informática
ibaeza@utem.cl

Javier Nanco
Departamento de Ingeniería Informática
jnanco@utem.cl

14 de julio de 2024

Índice

1. Resumen

Este documento aborda el problema de determinar el número mínimo de cortes necesarios para particionar una cadena de caracteres en subcadenas que sean palíndromos. Se exploran dos métodos principales: uno basado en programación dinámica que optimiza subproblemas superpuestos con una complejidad temporal de $O(n^2)$ y otro que utiliza un enfoque recursivo con una complejidad de $O(n^3)$. Ambos métodos emplean matrices 2D para almacenar los resultados computados y verificar si las subcadenas son palíndromas.

El estudio se enfoca en implementar estos algoritmos, medir su tiempo de ejecución con cadenas de distintos tamaños, analizar su complejidad computacional y comparar sus rendimientos mediante gráficos. El objetivo es optimizar el proceso de particionamiento de palíndromos y proporcionar una solución eficaz para aplicaciones que requieren procesamiento intensivo de texto.

2. Introduccion

Este documento presenta un análisis exhaustivo de diversos métodos computacionales para abordar problemas específicos en el campo de la programación, enfocándose particularmente en la tecnica de Programacion Dinamica aplicada tanto de manera cuadratica como cubica. A través de un enfoque metodológico riguroso, se busca no solo identificar las soluciones más eficientes, sino también explorar las limitaciones y desafíos inherentes a estos métodos. La investigación se contextualiza dentro de un marco teórico que comprende un detallado experimento gráfico para ilustrar las diferencias de rendimiento entre los enfoques analizados.

El documento está estructurado para guiar al lector a través de una comprensión completa de la problemática abordada. Se inicia con un resumen y una introducción detallada, seguido de una descripción del problema específico que se aborda. La metodología aplicada para enfrentar el problema incluye Programación Dinamica apliacada tanto de manera cuadratica como cúbica con un enfoque particular en cómo estas técnicas pueden ser aplicadas para resolver eficazmente los problemas propuestos.

El núcleo del estudio radica en la problemática de optimizar algoritmos para tareas específicas, demostrando cómo la Programación Dinámica tanto cuadratica como cubica ofrecen soluciones viables pero con diferentes implicaciones en términos de eficiencia y complejidad. La sección dedicada a la solución desglosa estos métodos en sus respectivos subcomponentes, evaluando sus méritos y limitaciones en situaciones de uso real.

En términos metodológicos, el documento detalla la planificación y ejecución de experimentos que incluyen pruebas de rendimiento comparativo visualizadas a través de gráficos. Este enfoque no solo clarifica las diferencias entre las estrategias de solución, sino que también ofrece una perspectiva visual de su eficacia relativa frente a diversas condiciones de prueba.

Finalmente, se explicara completa el entorno técnico utilizado para la investigación y ejecución, incluyendo especificaciones detalladas del hardware y el software, lo que permite una comprensión profunda de las condiciones bajo las cuales se realizaron los experimentos. Esta sección es fundamental para aquellos interesados en replicar o extender la investigación realizada.

Este documento, por lo tanto, no solo sirve como un registro de investigación científica, sino también como un manual práctico para la implementación de soluciones computacionales avanzadas en problemas reales.

3. Problema

La problemática del estudio se centra en optimizar el proceso de particionamiento de palíndromos, una tarea computacionalmente intensiva que busca minimizar los cortes necesarios para descomponer una cadena en subcadenas palíndromas. Este problema tiene implicaciones significativas en áreas como procesamiento de texto, donde la eficiencia y la precisión del algoritmo son críticas. La elección y optimización del algoritmo adecuado son cruciales para manejar cadenas grandes eficientemente, representando un desafío tanto en términos de complejidad computacional como de tiempo de ejecución.

3.1. Metodologia

En la investigación se emplean métodos de programación dinámica para abordar el problema del particionamiento de palíndromos. Se comparan dos enfoques diferentes: uno cuadrático y otro cúbico, utilizando diversas cadenas de texto como datos de entrada.

3.2. Solución

La solución propuesta se basa en la implementación de dos algoritmos de programación dinámica con distintas complejidades computacionales. Se diseñaron pruebas para medir y comparar el rendimiento y la eficiencia de cada algoritmo.

3.2.1. Programación dinamica (Cuadratica)

Este enfoque emplea un algoritmo que itera sobre cada posible subcadena de la cadena original, comprobando si es un palíndromo y actualizando el mínimo número de cortes necesarios para dividir la cadena en palíndromos. En términos de código, el algoritmo inicializa un arreglo cortesminimosdp donde cada índice final representa el mínimo de cortes necesarios hasta esa posición. Por cada posición final, se explora cada posición inicio anterior para determinar si la subcadena desde inicio hasta final es un palíndromo. Si es así, el arreglo se actualiza para reflejar el mínimo de cortes necesarios considerando un nuevo corte antes de inicio, si esa opción reduce el número de cortes previamente calculado.

3.2.2. Programación dinamica (Cubica)

Este método incrementa la complejidad al considerar todas las posibles maneras de dividir cada subcadena en dos, evaluando cada posible punto de división. Utiliza dos matrices: palindromo y mincut. La matriz palindromo se utiliza para memorizar subcadenas que son palíndromos, reduciendo la necesidad de recomputar este hecho para subcadenas repetidamente. La matriz mincut almacena el mínimo número de cortes necesarios para cada subcadena. Si la subcadena actual no es un palíndromo, el algoritmo busca el punto óptimo para

dividirla, minimizando los cortes totales. Esto se logra comparando los costos de cortar en cada posible punto y sumando un corte adicional para esa división.

3.3. Problemas

Durante las pruebas, se identificaron variaciones en el rendimiento dependiendo del modo de operación (con o sin ahorro de energía). Esto subraya la importancia de considerar el consumo de recursos en la implementación de algoritmos en dispositivos con capacidad limitada.

3.4. Experimento grafico

A continuación se presentan dos diagramas que representan el rendimiento de dos algoritmos distintos. El primero de estos algoritmos, representado mediante una línea de color rojo, corresponde al algoritmo de programación dinámica cúbica; mientras que el segundo, representado por una línea de color azul, se refiere al algoritmo de programación dinámica cuadrática. Adicionalmente, se incluyen dos gráficos en la representación: el gráfico superior ilustra los resultados con el ahorro de energía activado, y el gráfico inferior muestra los resultados sin el ahorro de energía. Cabe destacar que esta diferenciación fue establecida tras realizar diversas pruebas, lo que nos permitió identificar estas problemáticas específicas.

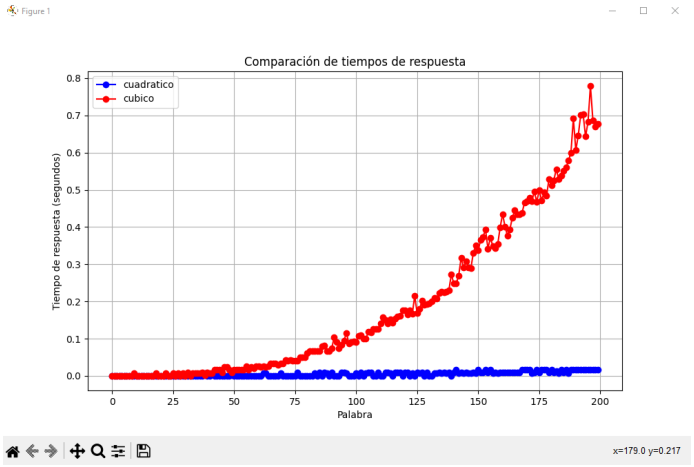

Figura 1: Comparativa gráfica entre el método cuadrático y cúbico (Ahorro de energía activado).

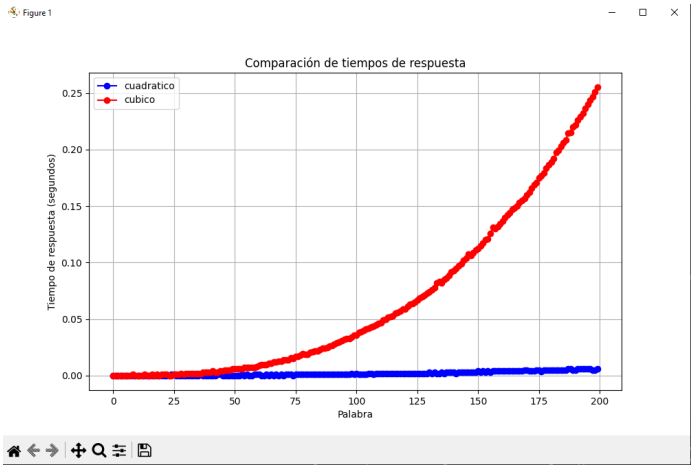

Figura 2: Comparativa gráfica entre el método cuadrático y cúbico (Ahorro de energía desactivado).

4. Conclusiones

En este estudio, se evaluaron dos metodologías de programación dinámica, cuadrática y cúbica, para resolver el problema de particionamiento de palíndromos en cadenas de texto, demostrando diferencias significativas en su rendimiento y eficiencia. La implementación cuadrática mostró mayor eficiencia en escenarios de menor complejidad, siendo óptima para aplicaciones que demandan rapidez en entornos con recursos limitados, mientras que la versión cúbica fue capaz de manejar casos más complejos a expensas de un mayor tiempo de procesamiento. Los resultados, visualizados a través de gráficos comparativos, subrayan la importancia de la elección del algoritmo en función del contexto operativo y la escala del problema.

5. Apendice

5.1. Codigo

5.1.1. Importaciones

```
import time
import random
import matplotlib.pyplot as plt
```

5.1.2. Metodo por Programacion Dinamica (Cuadratico)

```
def min_cut_dp(largo,cortesminimosdp,palabra):
    start_time = time.time()

    #inicio
    for final in range(1, largo):
        cortesminimosdp[final] = final

        for inicio in range(final, -1, -1):
            if palabra[inicio:final+1] == palabra[inicio:final+1][::-1]:
                if inicio == 0:
                    cortesminimosdp[final] = 0
                else:
                    cortesminimosdp[final] = min(cortesminimosdp[final],
                    # Salto de linea
                    # No presente en
                    # El codigo original
                    1 + cortesminimosdp[inicio - 1])


    end_time = time.time()
    return end_time - start_time
```

5.1.3. Metodo por Programacion Dinamica (Cúbico)

```python
def min_palindrome_cuts(mincut,palindromo,largo,string):
    start_time = time.time()

    # Calcular para subcadenas más grandes
    for largo_interno in range(2, largo + 1):
        for i in range(largo - largo_interno + 1):
            final = i + largo_interno - 1
            if largo_interno == 2:
                palindromo[i][final] = (string[i] == string[final])
            else:
                palindromo[i][final] = (string[i] == string[final])
                # Salto de linea
                # No presente en
                # El codigo original
                and palindromo[i + 1][final - 1]

            if palindromo[i][final]:
                mincut[i][final] = 0
            else:
                mincut[i][final] = float('inf')
                for k in range(i, final):
                    mincut[i][final] = min(mincut[i][final]
                    # Salto de linea
                    # No presente en
                    # El codigo original
                    , mincut[i][k] + mincut[k + 1][final] + 1)

    end_time = time.time()
    return end_time - start_time
```

5.1.4. Inicializadores

```python
def inicializar_cuadratico(palabra):
    largo = len(palabra)
    cortesminimosdp = [0] * largo
    return min_cut_dp(largo,cortesminimosdp,palabra)

def inicializar_cubico(string):
    n = len(string)
    C = [[0] * n for _ in range(n)]
    P = [[False] * n for _ in range(n)]

    # Casos base (diagonales)
    for i in range(n):
```

```
            P[i][i] = True

    return min_palindrome_cuts(C,P,n,string)
```

5.1.5. Main y revision del tiempo

```

def generar_palabras(num_palabras):
    palabras = []
    for i in range(5,num_palabras+5):
        palabra = ''.join(random.choices('abcde', k=i))
        palabras.append(palabra)
    return palabras

# visualización de los tiempos de respuesta con un grafico bonito c:
def visualizar_tiempos(tiempos_A, tiempos_B):
    plt.figure(figsize=(10, 6))
    plt.plot(range(len(tiempos_A)), tiempos_A,
    # Salto de linea
    # No presente en
    # El codigo original
    color='blue', marker='o', label='cuadratico')
    plt.plot(range(len(tiempos_B)), tiempos_B,
    # Salto de linea
    # No presente en
    # El codigo original
    color='red', marker='o', label='cubico')
    plt.xlabel('Palabra')
    plt.ylabel('Tiempo de respuesta (segundos)')
    plt.title('Comparación de tiempos de respuesta')
    plt.legend()
    plt.grid(True)
    plt.show()

# medicion de tiempos en base a las palabras
def medir_tiempos(palabras):
    tiempos_A = []
    tiempos_B = []
    for palabra in palabras:
        tiempo_A = inicializar_cuadratico(palabra)
        tiempo_B = inicializar_cubico(palabra)
        tiempos_A.append(tiempo_A)
        tiempos_B.append(tiempo_B)
    return tiempos_A, tiempos_B

cantidad_letras = 200
```

```
41
42  palabras = generar_palabras(cantidad_letras)
43
44  tiempos_A, tiempos_B = medir_tiempos(palabras)
45
46  visualizar_tiempos(tiempos_A, tiempos_B)
```

5.2. Instrucciones para la ejecución del programa

En el ámbito de la investigación científica, la elección de herramientas de programación y visualización de datos es crucial para la eficiencia y efectividad del análisis. Python es ampliamente utilizado debido a su sintaxis intuitiva, que facilita la escritura de código legible y mantenible, su extenso ecosistema de bibliotecas robustas, y su gran comunidad de desarrolladores que ofrece soporte continuo y desarrollo constante. Estas características hacen de Python una opción ideal para aplicaciones de investigación que requieren adaptabilidad y escalabilidad.

Por otro lado, Matplotlib es una biblioteca de visualización en Python que se utiliza por su capacidad para generar una amplia variedad de gráficos y visualizaciones estáticas, animadas e interactivas. Esta biblioteca es esencial para la interpretación visual de datos complejos, permitiendo a los investigadores presentar sus resultados de manera clara y comprensible. La flexibilidad de Matplotlib en la personalización de gráficos asegura que las visualizaciones puedan ser ajustadas para satisfacer las necesidades específicas de cualquier estudio, apoyando la reproducibilidad y la precisión en la comunicación de hallazgos científicos.

5.2.1. Instalación en Windows

Paso 1: Instalación de Python

1. **Descarga de Python**:
   - Visita el sitio web oficial de Python en https://www.python.org/.
   - Navega hasta la sección “Downloads” y selecciona “Windows”.
   - Haz clic en el enlace “Download Python” seguido del número de la última versión (por ejemplo, Python 3.9).
2. **Ejecución del Instalador**:
   - Una vez descargado, abre el instalador de Python.
   - Muy importante: Asegúrate de marcar la opción “Add Python 3.x to PATH” al inicio del proceso de instalación para incluir Python en la variable de entorno PATH de Windows.
   - Haz clic en “Install Now” para iniciar la instalación con la configuración recomendada.

3. **Verificación de la Instalación**:

   - Para verificar que Python está correctamente instalado, abre el Command Prompt (CMD) y escribe:

     ```
     python --version
     ```

   - Deberías ver el número de la versión de Python que se acaba de instalar.

Paso 2: Instalación de Matplotlib

1. **Apertura de CMD**:

   - Asegúrate de que la línea de comando (CMD) esté abierta para realizar la instalación de Matplotlib.

2. **Instalación de Matplotlib**:

   - En el CMD, escribe el siguiente comando para instalar Matplotlib usando pip, el gestor de paquetes de Python:

     ```
     pip install matplotlib
     ```

5.2.2. Instalación en Linux

Paso 1: Instalación de Python

1. **Actualizar el Gestor de Paquetes**:

   - Abre una terminal.
   - Actualiza el gestor de paquetes de tu distribución Linux con uno de los siguientes comandos, dependiendo de tu sistema:

     ```
     # para Debian y Ubuntu
     sudo apt update && sudo apt upgrade
     # para Fedora
     sudo yum update
     # para Arch Linux
     sudo pacman -Syu
     ```

2. **Instalación de Python**:

   - La mayoría de las distribuciones de Linux vienen con Python preinstalado. Para instalar o verificar si Python está instalado, usa el comando correspondiente a tu distribución:

```
sudo apt install python3  # Para Debian/Ubuntu
sudo yum install python3  # Para Fedora
sudo pacman -S python3  # Para Arch Linux
```

3. **Verificación de la Instalación**:
   - Verifica que Python está correctamente instalado ejecutando:

     ```
     python3 --version
     ```

   - Esto debería mostrar la versión de Python instalada.

Paso 2: Instalación de Matplotlib

1. **Instalar pip**:
   - `pip` es el gestor de paquetes de Python y puede necesitar ser instalado:

     ```
     sudo apt install python3-pip  # Debian/Ubuntu
     sudo yum install python3-pip  # Fedora
     sudo pacman -S python-pip  # Arch Linux
     ```

2. **Instalación de Matplotlib**:
   - Con `pip` instalado, puedes proceder a instalar Matplotlib utilizando:

     ```
     pip3 install matplotlib
     ```

5.2.3. Instalación en Mac

Paso 1: Instalación de Python

1. **Utilizar Homebrew**:
   - Homebrew es un gestor de paquetes para macOS que facilita la instalación de software. Si no tienes Homebrew instalado, puedes instalarlo ejecutando el siguiente comando en la terminal:

     ```
     /bin/bash -c "$(curl -fsSL \url{https://raw.git
     hubusercontent.com/Homebrew/install/HEAD/instal
     l.sh})"
     ```

   - Asegúrate de seguir las instrucciones en pantalla para completar la instalación de Homebrew.

2. **Instalar Python con Homebrew**:

- Una vez que Homebrew está instalado, puedes instalar Python ejecutando:

```
brew install python
```

3. **Verificación de la Instalación**:

   - Verifica que Python está correctamente instalado ejecutando:

```
python3 --version
```

   - Esto debería mostrar la versión de Python instalada, indicando que la instalación fue exitosa.

Paso 2: Instalación de Matplotlib

1. **Instalar pip**:

   - `pip` viene instalado con Python cuando se instala a través de Homebrew. Sin embargo, si necesitas asegurarte de que está actualizado, puedes ejecutar:

```
python3 -m pip install --upgrade pip
```

2. **Instalación de Matplotlib**:

   - Con `pip` listo, puedes instalar Matplotlib utilizando:

```
pip3 install matplotlib
```

5.3. Especificaciones técnicas del sistema informático empleado

En el desarrollo del proyecto, se seleccionaron componentes de hardware que cumplen con los requerimientos necesarios para garantizar un rendimiento óptimo. Específicamente se indicara las configuraciones a continuación.

5.3.1. CPU

Para garantizar un rendimiento robusto y eficiente, se seleccionó un procesador de alto rendimiento con las siguientes características:

- **Modelo:** AMD Ryzen 7 4800H
- **Arquitectura:** x86-64 (64 bits)
- **Frecuencia base:** 2.9 GHz
- **Núcleos:** 8
- **Hilos:** 16

5.3.2. Memoria RAM

Para el sistema, se integró un módulo de memoria RAM con una capacidad total de 8 GB, lo que permite manejar eficientemente las operaciones y procesos exigidos por el software utilizado. La memoria seleccionada corresponde al tipo DDR4, conocida por su alta tasa de transferencia y eficiencia energética. Las especificaciones técnicas son las siguientes:

- **Tipo:** DDR4
- **Capacidad total:** 8 GB
- **Velocidad de transferencia:** 2400 MHz

5.3.3. Unidad de Procesamiento Gráfico (GPU)

Para garantizar un rendimiento gráfico adecuado y eficiente en el manejo de aplicaciones gráficas y de procesamiento paralelo, se seleccionó la tarjeta gráfica NVIDIA GeForce GTX 1650. Las especificaciones técnicas de la GPU son las siguientes:

- **Modelo:** NVIDIA GeForce GTX 1650
- **Arquitectura:** Turing
- **Memoria VRAM:** 4 GB GDDR5
- **Velocidad de la memoria:** 8 Gbps

5.4. Entorno de desarrollo

Para el desarrollo de este proyecto, utilizamos la última versión del editor de código Visual Studio Code (VS Code), junto con las extensiones necesarias para trabajar con Python. VS Code se seleccionó por su flexibilidad, facilidad de uso y la amplia gama de herramientas y extensiones disponibles que facilitan el desarrollo de aplicaciones complejas.

5.5. Visual Studio Code (VS Code)

VS Code es un editor de código fuente gratuito y potente que soporta múltiples lenguajes de programación. En nuestro caso, se utiliza principalmente para el desarrollo en Python. Las características clave de VS Code que beneficiaron nuestro proyecto son:

- **Flexibilidad y Extensibilidad:** Permite la instalación de extensiones que amplían sus capacidades.
- **Depuración Integrada:** Facilita la identificación y corrección de errores en el código.
- **Control de Versiones:** Integración nativa con sistemas como Git, facilitando la gestión del código fuente.
- **Entorno de Desarrollo Unificado:** Combina editor, depurador y terminal en una única interfaz.

5.6. Extensiones para Python

Para optimizar el desarrollo en Python, se instalaron y configuraron varias extensiones específicas, entre las cuales se destacan:

- **Python (Microsoft):** Proporciona soporte para la sintaxis de Python, depuración, IntelliSense y otras herramientas avanzadas.
- **Pylint:** Extensión para análisis estático del código, que ayuda a mantener un código limpio y libre de errores.
- **Copilot** Extensión principalmente utilizada para autosugerencias al momento de la creacion de codigo repetitivo.

Estas herramientas y extensiones permiten un desarrollo más eficiente y ayudan a garantizar que el código cumpla con los estándares de calidad requeridos.

6. Referencias

Referencias

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd). MIT Press.

Luis H. Herrera. (2023). *Análisis y Desarrollo de Algoritmos*. https://utem.instru%20cture.com/courses/15665/files/366472?module_item_id=154307

Overleaf. (2024). Overleaf - Online LaTeX Editor [Accessed: 2024-07-14]. https://www.overleaf.com/read/cknnhmhsxbkz#4c9086