Programación Avanzada
Trabajo Practico
Complejidad Computacional + Programación Dinámica

2º cuatrimestre 2023

Universidad Nacional de La Matanza

1. Introducción

A través del presente trabajo se espera que los alumnos codifiquen, evalúen y comparen distintos algoritmos que resuelven, mediante diferentes técnicas, el mismo problema.

2. Consigna

2.1. Polinomios

Diseñar un programa para evaluar un polinomio $p(x)$ de grado n:

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n$$

Para ello, utilizaremos una estructura jerárquica polimórfica, donde deberán crear clases que implementen diversas estrategias para evaluar el polinomio en un valor de x.

```
public abstract class Polinomio {
    private int grado;
    private double[] coeficientes;

    public Polinomio(double[] coeficientes){...}

    public abstract double evaluar(double x);
    public String toString() {...}
}
```

Se sugiere que haya una correspondencia entre el grado del término del polinomio y el subíndice del array utilizado para almacenarlo, de modo que:

$$\begin{aligned} coeficientes[0] &= a_0 \\ coeficientes[1] &= a_1 \\ &\vdots \\ coeficientes[n-1] &= a_{n-1} \\ coeficientes[n] &= a_n \end{aligned}$$

2.2. Implementaciones

A continuación se presentan las implementaciones que deberán ser codificadas, probadas y comparadas.

2.2.1. Evaluación por Multiplicaciones Sucesivas

Utilizará una estrategia de multiplicaciones sucesivas para el cálculo de las potencias de x.

2.2.2. Evaluación por Recursividad

Utilizará una estrategia recursiva para el cálculo de las potencias de x.

2.2.3. Evaluación por Recursividad Par/Impar

Utilizará una estrategia recursiva para el cálculo de las potencias de x, considerando si el exponente es par o impar, de la siguiente manera:

$$pow(x,n) = \begin{cases} pow(x \cdot x, n/2) & \text{si } n \text{ mód } 2 == 0 \\ x \cdot pow(x, n-1) & \text{caso contrario} \end{cases}$$

2.2.4. Evaluación Iterativa Par/Impar

Utilizará la misma estrategia que la implementación anterior, pero evitando utilizar la aproximación recursiva.

2.2.5. Evaluación por Programación Dinámica

Utilizará una estrategia que reutilice el cálculo de las potencias de x.

2.2.6. Evaluación por Estrategia Mejorada

Utilizará una estrategia de igual complejidad que el código anterior, pero que arroje un mejor tiempo de ejecución.

2.2.7. Evaluación por Implementación Nativa de Math.pow

Utilizará la implementación por defecto de Math.pow. Investigar la complejidad computacional de dicha implementación.

2.2.8. Evaluación por Método de Horner

Utilizará el Método de Horner para la evaluación de polinomios, de la siguiente manera:

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n = a_0 + x\Big(a_1 + x\Big(a_2 + x\big(a_3 + \cdots + x(a_{n-1} + x\, a_n)\cdots\big)\Big)\Big)$$

3. Análisis de complejidad computacional

Indique la función de complejidadad computacional asociada a cada una de las estrategias implementadas. Justificar mediante un análisis del código, sentencia por sentencia.

4. Gráficos y tablas de rendimiento comparativo

Compare el tiempo de ejecución de todas las estrategias implementadas. Deberá llevar a cabo una estrategia que le permita tomar comparaciones significativas y cuya afectación por causas externas al algoritmo sea despreciable.

Genere todos los casos que considere necesarios para realizar el análisis.

5. Conclusiones

A partir de los análisis comparativos extraiga conclusiones. Plantear las mismas en formato de preguntas y respuestas.