Juego de adivinar un número ⋆

Natalia Mejía Estrada[a]

[a] *Pontificia Universidad Javeriana, Bogotá, Colombia*

Abstract

En este documento se presenta la escritura formal del problema «Juego de adivinar un número» junto con un algoritmo de solución de tipo dividir-y-vencer.

Keywords: algoritmo, escritura formal, juego, adivinar número.

1. Análisis del problema

El juego de adivinar un numero se juega entre dos personas: una de ellas piensa en un número natural y la otra intenta adivinarlo dando algunas posibles respuestas. En cada turno, el adivinador da un conjunto de n números y el pensador indica para cada número, si es mayor, menor o igual.

2. Diseño del problema

El análisis anterior nos permite diseñar el problema: definir las entradas y salidas de un posible algoritmo de solución, que aun no esta definido.

1. *Entradas*: Número real k: $k \in \mathbb{R}$
   Límite inferior del intervalo b: $b < k$
   Límite superior del intervalo e: $e > k$ y $e > b$
   Categorización de los intentos de adivinar:
   - 0: Igual
   - 1: Menor
   - 2: Mayor
2. *Salidas*: La retroalimentación del pensador sobre cada suposición del adivinador es: Mayor // Menor // Igual
   El número adivinado por el adivinador, si es correcto, es una cadena de texto que representa el número adivinado (a) tal que $a = k$.

⋆Este documento presenta la escritura formal de un algoritmo.

Email address: nataliamejia@javeriana.edu.co (Natalia Mejía Estrada)

3. Algoritmos de solución

3.1. Algoritmo dividir-y-vencer

Este algoritmo de solución usa la estrategia de dividir y vencer para hallar la solución al problema.

Algorithm 1 Algoritmo usando dividir y vencer para jugar el juego de adivinar un numero.

Require: k: $k \in \mathbb{R}$
Require: b: $b < k$
Require: e: $e > k$ y $e > b$

```
1:  procedure ADIVINAR(k, b, e)
2:      if b <= e then
3:          q ← b + e/2
4:          response ← getGuessResponse (q)
5:          if response = 0 then
6:              return 0
7:          end if
8:          if response = 1 then
9:              return Adivinar (k, b, q − 1)
10:         end if
11:         if response = 2 then
12:             return Adivinar (k, q + 1, e)
13:         else
14:             return
15:         end if
16:     end if
17: end procedure
```

3.1.1. Invariante

Durante el algoritmo, se mantiene que el inicio b siempre es menor que el fin e y que el número pensado $k$estará dentro del rango especificado por inicio y fin. Adicionalmente, debido a la respuesta del pensador o *getGuessResponse* (mayor, menor o igual), el algoritmo puede tomar la decisión de si la respuesta es:
1 (menor), significa que el número pensado es menor que el número en el punto medio y actualiza el límite superior del intervalo (fin) al punto medio menos uno y continúa la búsqueda en la mitad inferior;
2 (mayor), significa que el número pensado es mayor que el número en el punto medio, entonces se actualiza el límite inferior del intervalo (inicio) al punto medio más uno y continúa la búsqueda en la mitad superior;
0(igual), significa que el número medio es igual al pensado y es correcto.

3.1.2. Análisis de complejidad

Como se puede notar, el algoritmo tiene 2 recurrencias, y adicionalmente el pivote se define dividiendo el intervalo entre 2. Esto significa que se tiene una

complejidad temporal de $T(n) = 2T(n \div 2) + O(1)$. Por teorema maestro, se sabe que esta complejidad corresponde a $\theta(n)$ igual a los previamente hechos en clase.

3.1.3. Notas de implementación

Para las entradas, se debe verificar que el numero a adivinar este dentro del intervalo, que el intervalo sea coherente al numero a adivinar y la respuesta a cada adivinación introducida por el usuario, corresponda a uno de los 3 valores. Dentro del procedimiento no es necesario el uso de k pero se pone para hacer comprobaciones frente al intervalo.