Introducción a las Ciencias Computacionales y Programación con Rust
Lectura 0

Gustavo Santos

June 1, 2025

Abstract

Las ciencias computacionales en 2025 han adquirido una relevancia creciente debido a su aplicación en áreas como la inteligencia artificial, la ciberseguridad y el desarrollo de software avanzado. Este trabajo ofrece una introducción a los fundamentos actuales de la computación, analizando su impacto en diversos sectores y destacando las habilidades clave requeridas para quienes desean iniciar una carrera en este campo. El objetivo principal es proporcionar una base sólida para comprender los desafíos y oportunidades que presenta esta disciplina en la actualidad.

1 Introduccion

Las ciencias computacionales han transformado profundamente la manera en que vivimos, trabajamos y nos comunicamos. Su impacto es tan significativo que hoy en día resulta difícil imaginar un mundo sin su influencia. En este contexto, aprender sobre esta disciplina se ha vuelto una necesidad para quienes desean participar activamente en el desarrollo tecnológico.

Este trabajo tiene como objetivo ofrecer una introducción accesible a las ciencias computacionales, enfocándose particularmente en el lenguaje de programación Rust. Rust es uno de los lenguajes más innovadores y prometedores de la actualidad, gracias a su enfoque en la seguridad, el rendimiento y la concurrencia. Está cambiando la forma en que se desarrolla software moderno y es cada vez más adoptado en proyectos de alto impacto.

A lo largo de este paper, exploraremos los fundamentos de Rust, su relevancia en la programación contemporánea, y cómo puede servir como una puerta de entrada para comprender conceptos clave en la computación.

2 Resolucion de problemas

2.1 Diagrama de caja negra

En esencia, la programación informática consiste en resolver problemas abstractos. Esto implica definir un conjunto finito de pasos —un algoritmo— que transforma los datos iniciales del problema, conocidos como **entrada** (*input*), en una **solución** o **salida** (*output*).

paper En el curso de CS50 de Harvard, a este proceso que ocurre entre la entrada y la salida se le denomina *caja negra* (*black box*). En este curso pondremos especial énfasis en comprender y construir esa caja negra, es decir, los algoritmos y programas que resuelven los problemas.

3 Sistema Binario

Actualmente, las computadoras modernas utilizan el sistema binario. Este sistema de numeración está basado en dos dígitos: 0 y 1. Es fundamental porque las computadoras funcionan gracias a los transistores, que son dispositivos electrónicos semiconductores capaces de modificar una señal eléctrica de salida en respuesta a una señal de entrada.

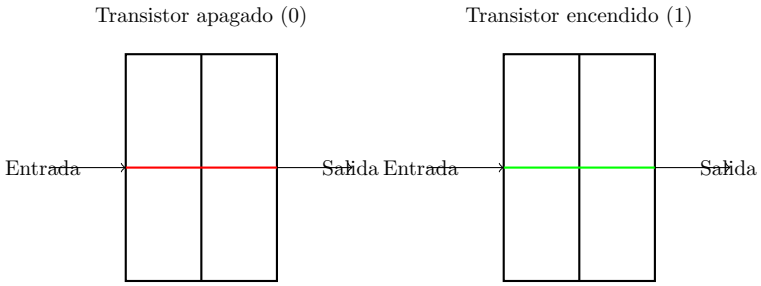
Los transistores funcionan controlando el paso de corriente eléctrica: pueden estar "encendidos" (permitiendo el paso de energía) o "apagados" (bloqueando el paso). El sistema binario se adapta perfectamente a esta característica, asignando el valor 1 para el estado encendido y el valor 0 para el estado apagado.

Los CPU modernos pueden contener miles de millones de transistores, lo que les permite realizar cálculos extremadamente complejos y rápidos.

Ejemplo básico de números en sistema binario:

Decimal	Binario
0	0
1	1
2	10
3	11
4	100
5	101

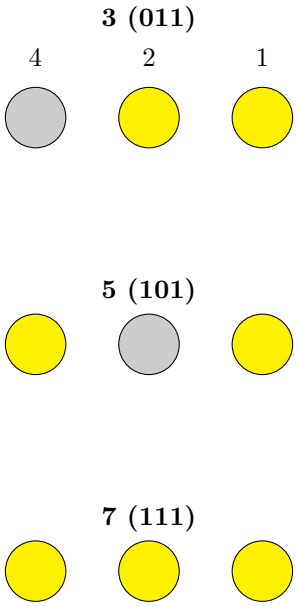
Diagrama simple de transistor en estado apagado y encendido:



Para que una computadora pueda contar, necesita usar el sistema binario. Imagina que tienes tres focos; cada uno puede estar encendido (representando un 1) o apagado (representando un 0).

Estos tres focos representan posiciones de valor 4, 2 y 1, respectivamente (de izquierda a derecha). Por ejemplo:

- El número 3 en binario se representa como `011` (foco 2 y foco 1 encendidos, foco 4 apagado).
- El número 5 en binario se representa como `101` (foco 4 y foco 1 encendidos, foco 2 apagado).
- Con tres focos, el número máximo que se puede representar es 7, que en binario es `111` (todos los focos encendidos).



Cada dígito binario, ya sea un 0 o un 1, se conoce como un **bit**. Generalmente, las computadoras utilizan un sistema de 8 bits para representar números. A este conjunto de 8 bits se le llama un **byte**.

Cada bit en un byte representa una potencia de 2, acomodada de la siguiente manera (de izquierda a derecha):

$$128 \quad 64 \quad 32 \quad 16 \quad 8 \quad 4 \quad 2 \quad 1$$

Por ejemplo, para representar el número 77 en binario, buscamos qué combinación de esos valores suma 77:

$$77 = 64 + 8 + 4 + 1 \Rightarrow \texttt{01001101}$$

También podemos observar que:

- El número máximo que se puede representar con 8 bits es `11111111`, que equivale a 255.
- El número mínimo es `00000000`, que representa el 0.

4 ASCII

Para nosotros, como humanos, sería muy complicado intentar enviar o leer mensajes completos en binario; tomaría mucho tiempo. Sin embargo, una computadora puede procesar esta información extremadamente rápido.

Para solucionar este problema, se desarrolló un estándar llamado **ASCII** (por sus siglas en inglés: *American Standard Code for Information Interchange*), que es un código de caracteres basado en el alfabeto latino tal como se usa en el inglés moderno. Este sistema asigna un número a cada letra, símbolo o carácter especial.

Por ejemplo:

- La letra `A` tiene asignado el número `65`, que en binario es `01000001`.

Así, al enviar `01000001`, en realidad estamos representando la letra `A`.

Ejemplo práctico:

Si tienes los números:

$$72 \quad 73 \quad 33$$

Esto representa los siguientes caracteres en ASCII:

- `72` → `H` → `01001000`
- `73` → `I` → `01001001`
- `33` → `!` → `00100001`

Por lo tanto, la secuencia completa representa el mensaje: `HI!`

Dado que un byte (8 bits) solo puede representar valores entre `0` y `255`, estamos limitados a un total de **256 combinaciones posibles**. Por lo tanto, el sistema **ASCII estándar** solo puede representar hasta **128 caracteres**, utilizando valores del `0` al `127`. Esto incluye letras, números, signos de puntuación y algunos caracteres de control (como salto de línea o tabulación).

Posteriormente, se creó una extensión llamada **ASCII extendido**, que utiliza el rango completo de `0` a `255`, permitiendo representar algunos símbolos adicionales, como letras acentuadas y caracteres gráficos. Sin embargo, incluso con esta extensión, el número total de caracteres sigue siendo limitado.

Esta limitación llevó al desarrollo de codificaciones más amplias como **Unicode**, que permiten representar prácticamente todos los caracteres de todos los idiomas del mundo.

5 Unicode

Unicode es un estándar de codificación de caracteres diseñado para facilitar el tratamiento, transmisión y visualización de textos en una gran variedad de idiomas del mundo. A diferencia del sistema ASCII, que está limitado a 128 o 256 caracteres, Unicode permite representar más de un millón de caracteres únicos.

Este estándar logra esto aumentando la cantidad de bits utilizados para representar cada carácter, lo que permite codificar no solo letras y símbolos de muchos alfabetos, sino también signos matemáticos, caracteres especiales, símbolos técnicos e incluso **emojis**.

Gracias a Unicode, hoy en día es posible escribir y compartir contenido digital en prácticamente cualquier idioma, lo que lo convierte en una base fundamental de la comunicación moderna.

6 RGB

Para la representación de colores en computadoras se utiliza el modelo **RGB**, que significa *Red, Green, Blue* (rojo, verde y azul). Este modelo se basa en la combinación de estos tres colores primarios de luz, donde cada uno puede tener una intensidad representada por un número entre `0` y `255` (es decir, un byte).

Por ejemplo, si usamos los valores `72`, `73` y `33`, que antes vimos como el texto "HI!" en binario, un visor de imágenes los interpretaría como un color. Esa combinación específica podría representar un tono amarillento.

Cada conjunto de tres bytes (uno para rojo, otro para verde y otro para azul) define un **píxel**, es decir, un punto de color en una imagen digital. Las imágenes están compuestas por miles o millones de estos píxeles, cada uno con su propia combinación RGB.

Los videos son simplemente una secuencia de muchas imágenes (fotogramas) reproducidas rápidamente, similar al efecto de un *flipbook*. Además, **el sonido digital** también puede ser representado con bytes, donde cada número representa la amplitud de una onda en un instante de tiempo.

Gracias a estas representaciones digitales basadas en bytes, las computadoras pueden manejar imágenes, videos y música de manera eficiente.

7 Algorithms

7.1 ¿Qué es un algoritmo?

Un **algoritmo** es un conjunto de instrucciones **no ambiguas**, **ordenadas** y **finitas**, que se ejecutan paso a paso con el fin de resolver un problema específico. En esencia, un algoritmo describe el procedimiento lógico que debe seguirse para obtener una solución, partiendo de una entrada (input) y generando una salida (output).

En este caso, revisaremos dos tipos de algoritmos clásicos para ilustrar su utilidad:

- Cómo calcular la **raíz cuadrada** de un número usando un proceso iterativo.
- Cómo **buscar un apellido** en un directorio telefónico utilizando búsqueda binaria.

7.2 Ejemplo de algoritmo: cálculo de la raíz cuadrada

Supongamos que queremos encontrar la raíz cuadrada de un número x. La raíz cuadrada de x es un número y tal que $y \times y = x$.

Podemos aplicar el siguiente algoritmo para aproximarnos a la raíz cuadrada:

1. Comenzamos con una suposición inicial g
2. Si $g \times g$ es suficientemente cercano a x, detenemos el proceso y g es la respuesta aproximada
3. De lo contrario, calculamos una nueva suposición promediando g con $\frac{x}{g}$:

$$g_{\text{nuevo}} = \frac{g + \frac{x}{g}}{2}$$

4. Repetimos el proceso usando el nuevo valor de g hasta obtener una aproximación suficientemente precisa.

A continuación, se muestra un ejemplo numérico para calcular la raíz cuadrada de $x = 16$:

g	$g \times g$	x/g	Nuevo $g = \frac{g + x/g}{2}$
3	9	5.33	4.17
4.17	17.36	3.837	4.0035
4.0035	16.0277	3.997	4.000002

Después de solo tres iteraciones, obtenemos una excelente aproximación de que la raíz cuadrada de 16 es aproximadamente 4. Esto muestra cómo un algoritmo puede resolver un problema matemático mediante pasos repetitivos.

7.3 Ejemplo de algoritmo: búsqueda binaria en una guía telefónica

Imagina que tienes una guía telefónica física, ordenada alfabéticamente, y deseas encontrar un solo nombre —por ejemplo, "Rodriguez".

Una estrategia muy ineficiente sería revisar página por página desde el principio hasta que lo encuentres. Eso se llama **búsqueda lineal**, y aunque funciona, es lenta si el libro es muy grande.

Sin embargo, un algoritmo más eficiente es la **búsqueda binaria**, que se basa en dividir el problema por la mitad en cada paso:

1. Abre la guía por la mitad.
2. Si el nombre en esa página está antes de "Rodriguez", entonces busca en la segunda mitad del libro.
3. Si está después, busca en la primera mitad.
4. Repite el proceso dividiendo el segmento restante por la mitad hasta encontrar el nombre deseado.

Este algoritmo aprovecha que la lista está **ordenada**, lo que le permite descartar la mitad del libro en cada paso.

Ventajas

La búsqueda binaria tiene una eficiencia de orden $O(\log_2 n)$, lo que significa que aunque haya un millón de nombres, solo necesitarías alrededor de 20 pasos para encontrar cualquiera.

Este algoritmo se usa no solo en guías telefónicas, sino también en motores de búsqueda, bases de datos y estructuras de datos como arreglos ordenados.

8 Pseudocódigo

Este proceso de convertir las instrucciones en una forma escrita paso a paso, sin necesidad de usar un lenguaje de programación específico, se llama **pseudocódigo**. Es una herramienta fundamental en la programación, ya que nos permite pensar y estructurar nuestras soluciones de forma lógica antes de escribir el código real.

El pseudocódigo es similar al lenguaje humano, pero lo suficientemente estructurado para describir algoritmos de manera clara. A continuación, se muestra un ejemplo de pseudocódigo para la búsqueda binaria de un apellido en un directorio telefónico:

```
Inicio
  Definir nombre_buscado
  Definir lista_ordenada
  Definir inicio = 0
  Definir fin = longitud(lista_ordenada) - 1

  Mientras inicio <= fin hacer
    medio = (inicio + fin) / 2
    Si lista_ordenada[medio] es igual a nombre_buscado
      Imprimir "Nombre encontrado"
      Terminar
    Si lista_ordenada[medio] < nombre_buscado
      inicio = medio + 1
    Sino
      fin = medio - 1
  FinMientras

  Imprimir "Nombre no encontrado"
Fin
```

Este pseudocódigo nos permite visualizar claramente los pasos que tomaría un algoritmo de búsqueda binaria, facilitando así la comprensión antes de programarlo en un lenguaje como Rust, Python o C++.

9 ¿Qué nos espera?

En esta lectura vimos fundamentos básicos de la informática, necesarios para comprender cómo funcionan las computadoras desde un nivel bajo. A partir de la próxima lectura, comenzaremos con el lenguaje de programación **Rust**, lo cual implicará un aumento en el nivel de dificultad.

Nuestro primer programa en Rust será el clásico *Hola, mundo*, que se verá de la siguiente manera:

```
fn main() {
    println!("Hola, mundo!");
}
```

Este será el primer paso para comenzar a escribir código real utilizando Rust.

References

[1] Lecture 0 - CS50x 2025. (s/f). Harvard.edu. Recuperado el 31 de mayo de 2025, de https://cs50.harvard.edu/x/notes/0/

[2] Lecture 1: Welcome. (s/f). MIT OpenCourseWare. Recuperado el 1 de junio de 2025, de https://ocw.mit.edu/courses/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/resources/mit6_0001f16_lec1/

[3] Transistor - Concepto, tipos y cómo funciona. (s/f). Recuperado el 31 de mayo de 2025, de https://concepto.de/transistor/

[4] Wikipedia contributors. (s/f-a). *Algoritmo*. Wikipedia, The Free Encyclopedia. Recuperado de https://es.wikipedia.org/w/index.php?title=Algoritmo&oldid=167508056

[5] Wikipedia contributors. (s/f-b). *ASCII*. Wikipedia, The Free Encyclopedia. Recuperado de https://es.wikipedia.org/w/index.php?title=ASCII&oldid=167298353

[6] Wikipedia contributors. (s/f-c). *RGB*. Wikipedia, The Free Encyclopedia. Recuperado de https://es.wikipedia.org/w/index.php?title=RGB&oldid=165193188

[7] Wikipedia contributors. (s/f-d). *Transistor*. Wikipedia, The Free Encyclopedia. Recuperado de https://es.wikipedia.org/w/index.php?title=Transistor&oldid=166178010

[8] Wikipedia contributors. (s/f-e). *Unicode*. Wikipedia, The Free Encyclopedia. Recuperado de https://es.wikipedia.org/w/index.php?title=Unicode&oldid=167376173