INSTITUTO POLITECNICO NACIONAL “IPN”

ESCUELA SUPERIOR DE COMPUTO “ESCOM”

LANDEROS CORTES PEDRO JONAS

MACHINE LEARNING

PRACTICA 2

INDICE

INDICE DE FIGURAS

Introducción

Dentro del campo de machine learning existen diversas técnicas inspiradas en procesos naturales para resolver problemas de optimización y búsqueda de soluciones.

Una de ellas son los algoritmos genéticos, que imitan la evolución biológica mediante operaciones como selección, cruce y mutación. Estos algoritmos no requieren un conocimiento explícito de la solución óptima, sino que generan y mejoran poblaciones de candidatos a lo largo de múltiples generaciones.

El programa desarrollado en esta práctica representa una simulación de este enfoque, en donde cada individuo se representa como un “vector de genes” con valores entre 1

y 9, y se define como un ser perfecto aquel cuya secuencia genética alcanza el valor máximo en todos los genes (9 en todos sus valores).

El sistema a genera una población inicial aleatoria y en cada generación forma parejas las cuales crean descendencia mediante promedios de gen a gen y aplica mutaciones probabilísticas que introducen diversidad, y el proceso sigue hasta que aparece un individuo perfecto o se llegan al máximo de generaciones.

Esta practica permite comprender como los algoritmos genéticos, al igual que otros métodos de machine learning, dependen de la calidad de los datos iniciales, de las reglas de variación y del criterio de aptitud para la búsqueda de soluciones. El modelo presentado refleja los elementos esenciales de un sistema de aprendizaje evolutivo.

Desarrollo

Flujo del programa

El flujo básico consta de cuatro etapas:

1. **Inicialización de población**: Se crea una población inicial de un tamaño configurable (por defecto 100), y cada individuo tendrá:
    - Id único.
    - Vector de 20 genes con valores aleatorios entre 1 y 9.
    - Porcentaje de mutación propio entre 0.1 y 0.9.
    - Un campo padre (vacío por ser la primera generación) que más tarde servirá para evitar cruce entre “hermanos”.
2. **Apareamiento y reproducción:** En cada generación se forman parejas aleatorias y se evita el cruce entre hermanos evitando que se crucen parejas con la misma id de padres, y para cada pareja se calcula el promedio gen a gen y si el valor es decimal se generan dos hijos, uno con un redondeo hacia abajo y otro con un redondeo hacia arriba.
3. **Mutación:** Cada hijo sufre una única mutación puntual, se selecciona un índice del vector de genes al azar y con un valor llamado p_mejora se aumenta la probabilidad hacia un valor mayor al actual si es que no está en el máximo y la p_empeora se define la probabilidad de que el valor sea menor al actual.
4. **Evaluación de población actual:** En cada nueva generación se evalúa si existe un individuo perfecto (con todos sus genes al máximo valor), y si aparece se reporta y se detiene el flujo del programa, si no la nueva generación sustituye a la anterior y se registran los ids de los descendientes y se continua, y se muestra

al mejor individuo actual de la generación según la suma total de genes (siendo el máximo 180).

Ejecución del programa

A continuación, se muestran distintas ejecuciones del programa con un valor p_mejora de 0.89 y p_empeora de 0.1

```
Generación 14257 | Mejor suma de genes: 173/180 | Mejor ID: 1425722
Generación 14258 | Mejor suma de genes: 172/180 | Mejor ID: 1425846
Generación 14259 | Mejor suma de genes: 174/180 | Mejor ID: 1425990
Generación 14260 | Mejor suma de genes: 173/180 | Mejor ID: 1426006
Generación 14261 | Mejor suma de genes: 172/180 | Mejor ID: 1426160
Generación 14262 | Mejor suma de genes: 171/180 | Mejor ID: 1426220
Generación 14263 | Mejor suma de genes: 172/180 | Mejor ID: 1426372
Generación 14264 | Mejor suma de genes: 174/180 | Mejor ID: 1426402
Generación 14265 | Mejor suma de genes: 173/180 | Mejor ID: 1426592
Generación 14266 | Mejor suma de genes: 174/180 | Mejor ID: 1426642
Generación 14267 | Mejor suma de genes: 180/180 | Mejor ID: 1426724
¡Humano perfecto encontrado en Generación 14267!
ID: 1426724 | Padres: (1426642, 1426684)
```

Figure 1. valor p_mejora de 0.89 y p_empeora de 0.1

```
Generación 364 | Mejor suma de genes: 173/180 | Mejor ID: 36412
Generación 365 | Mejor suma de genes: 175/180 | Mejor ID: 36550
Generación 366 | Mejor suma de genes: 172/180 | Mejor ID: 36612
Generación 367 | Mejor suma de genes: 174/180 | Mejor ID: 36766
Generación 368 | Mejor suma de genes: 176/180 | Mejor ID: 36864
Generación 369 | Mejor suma de genes: 175/180 | Mejor ID: 36954
Generación 370 | Mejor suma de genes: 174/180 | Mejor ID: 37086
Generación 371 | Mejor suma de genes: 176/180 | Mejor ID: 37150
Generación 372 | Mejor suma de genes: 174/180 | Mejor ID: 37230
Generación 373 | Mejor suma de genes: 174/180 | Mejor ID: 37312
Generación 374 | Mejor suma de genes: 176/180 | Mejor ID: 37444
Generación 375 | Mejor suma de genes: 178/180 | Mejor ID: 37546
Generación 376 | Mejor suma de genes: 179/180 | Mejor ID: 37674
Generación 377 | Mejor suma de genes: 180/180 | Mejor ID: 37746
¡Humano perfecto encontrado en Generación 377!
ID: 37746 | Padres: (37674, 37696)
```

Figure 2. valor p_mejora de 0.89 y p_empeora de 0.1

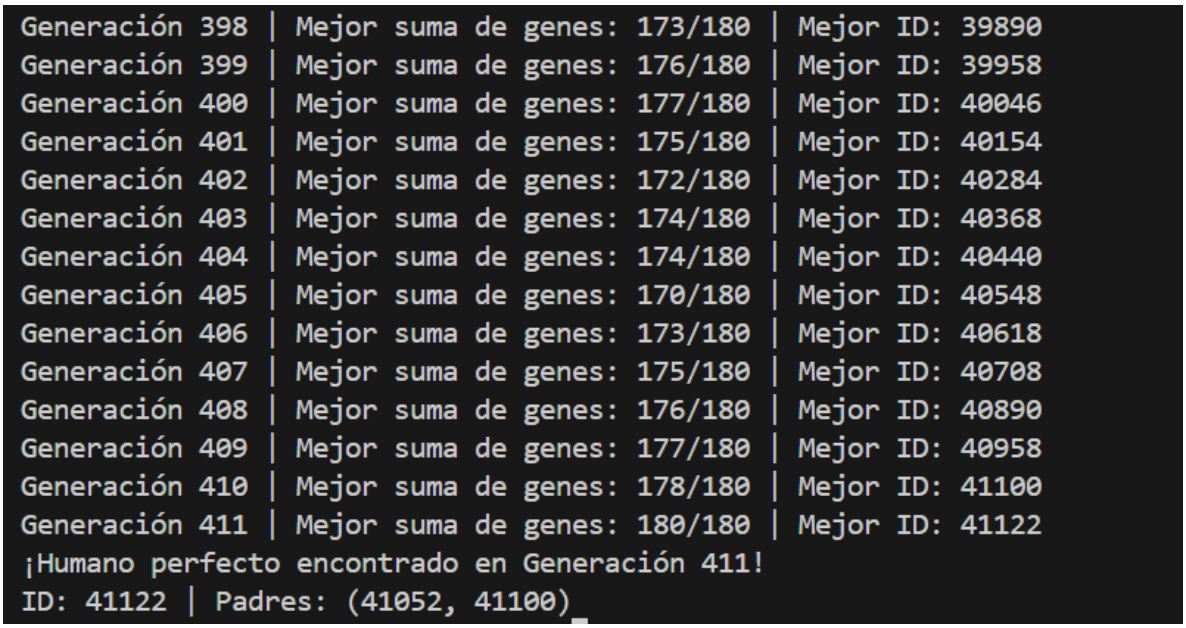

Figure 3. valor p_mejora de 0.89 y p_empeora de 0.1

Se puede observar que los valores varían entre cada ejecución, debido a que los valores iniciales son generados de manera aleatoria, lo que genera distintos valores en la generación del humano perfecto.

A continuación, se muestran distintas ejecuciones del programa con un valor p_mejora de 0.85 y p_empeora de 0.05

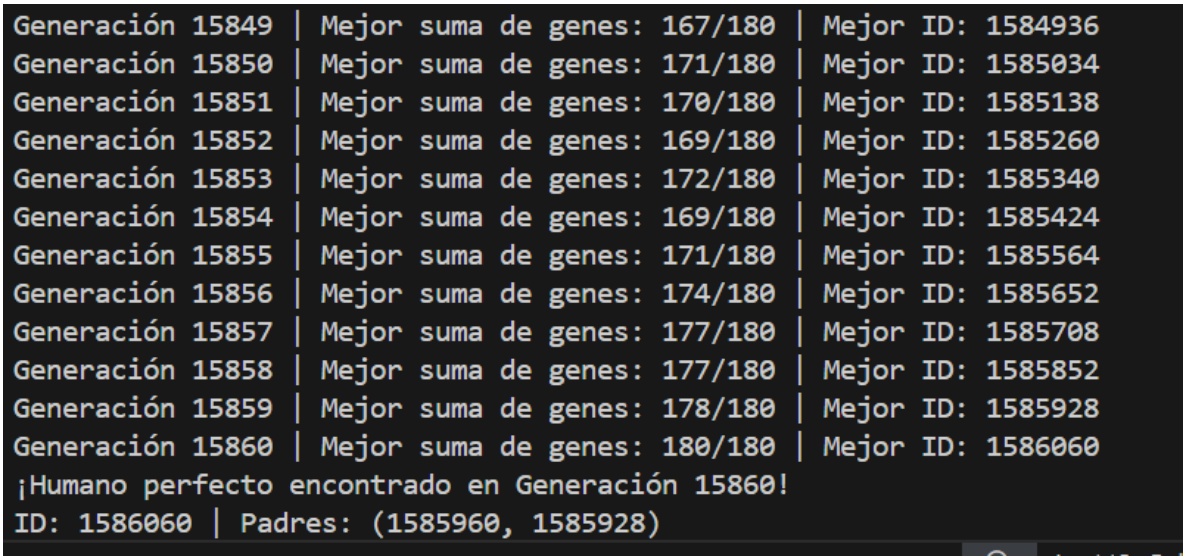

Figure 4. valor p_mejora de 0.85 y p_empeora de 0.05

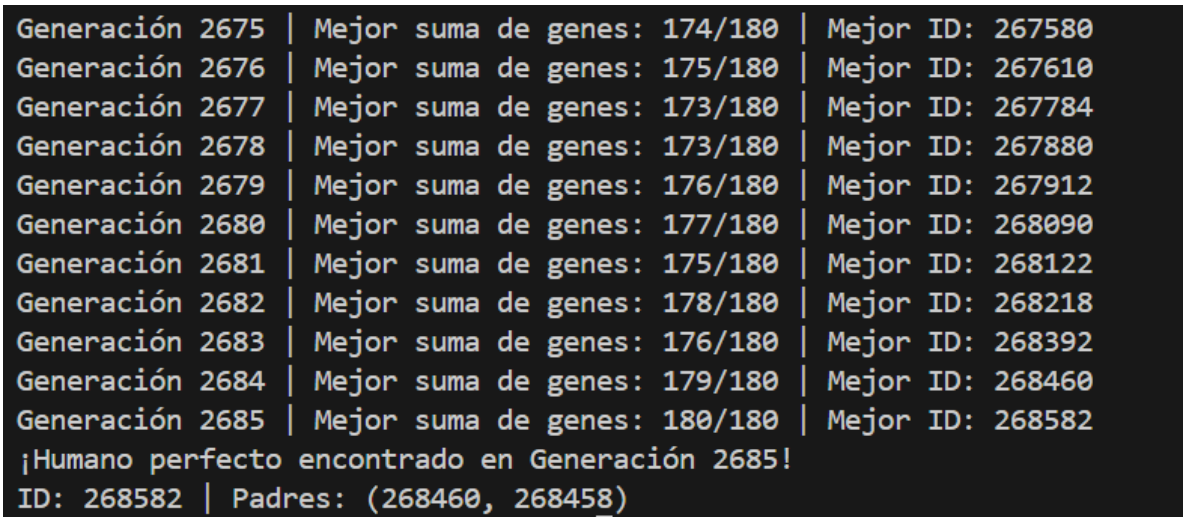

Figure 5. valor p_mejora de 0.85 y p_empeora de 0.05

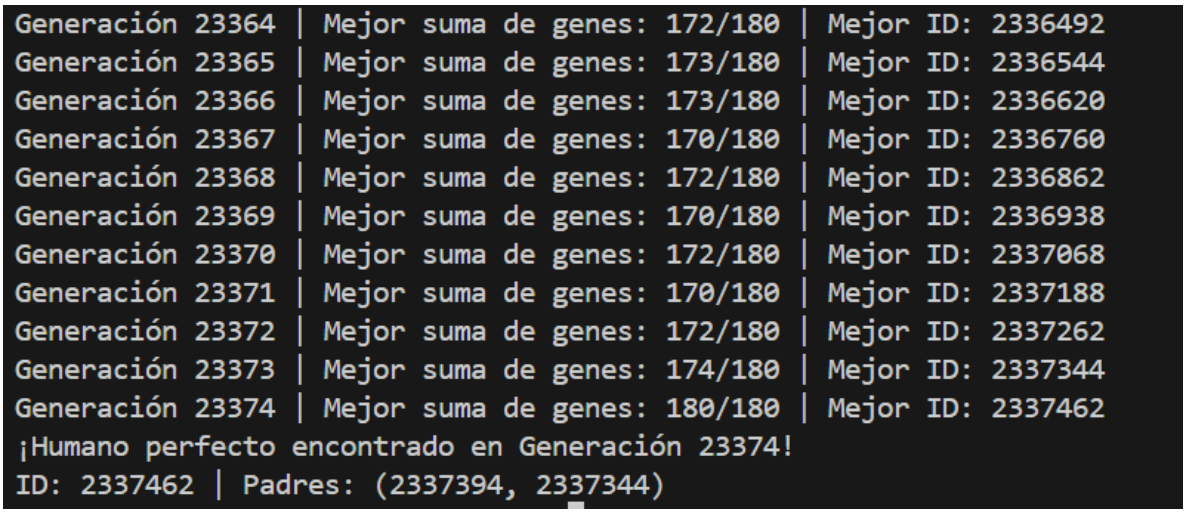

Figure 6. valor p_mejora de 0.85 y p_empeora de 0.05

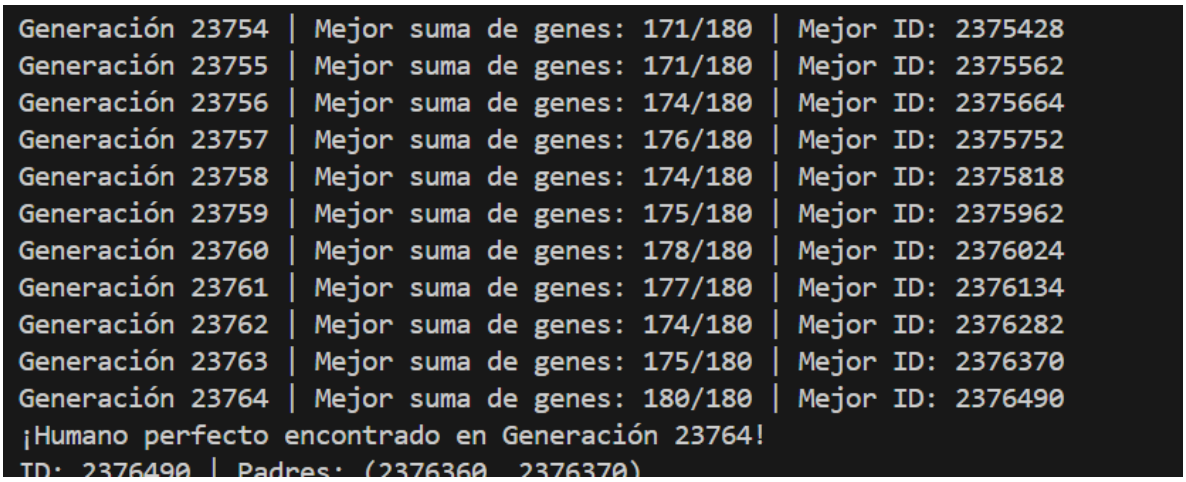

Figure 7. valor p_mejora de 0.85 y p_empeora de 0.05

Podemos observar que mientras mas se reduce la probabilidad de mejora de genes en cada vuelta y mas se aumenta (aunque sea por pocos valores) la probabilidad de empeorar, el numero de generaciones necesarias para encontrar al humano perfecto aumenta.

A continuación, se muestran distintas ejecuciones del programa con un valor p_mejora de 0.80 y p_empeora de 0.10

```
Generación 99989 | Mejor suma de genes: 168/180 | Mejor ID: 9999000
Generación 99990 | Mejor suma de genes: 165/180 | Mejor ID: 9999076
Generación 99991 | Mejor suma de genes: 164/180 | Mejor ID: 9999104
Generación 99992 | Mejor suma de genes: 167/180 | Mejor ID: 9999270
Generación 99993 | Mejor suma de genes: 169/180 | Mejor ID: 9999364
Generación 99994 | Mejor suma de genes: 169/180 | Mejor ID: 9999478
Generación 99995 | Mejor suma de genes: 169/180 | Mejor ID: 9999570
Generación 99996 | Mejor suma de genes: 170/180 | Mejor ID: 9999624
Generación 99997 | Mejor suma de genes: 168/180 | Mejor ID: 9999756
Generación 99998 | Mejor suma de genes: 170/180 | Mejor ID: 9999894
Generación 99999 | Mejor suma de genes: 167/180 | Mejor ID: 9999924
Generación 100000 | Mejor suma de genes: 169/180 | Mejor ID: 10000084
No se encontró un humano perfecto tras 100000 generaciones.
```

Figure 8. valor p_mejora de 0.80 y p_empeora de 0.10

```
Generación 99989 | Mejor suma de genes: 168/180 | Mejor ID: 9998966
Generación 99990 | Mejor suma de genes: 171/180 | Mejor ID: 9999002
Generación 99991 | Mejor suma de genes: 167/180 | Mejor ID: 9999180
Generación 99992 | Mejor suma de genes: 169/180 | Mejor ID: 9999254
Generación 99993 | Mejor suma de genes: 167/180 | Mejor ID: 9999314
Generación 99994 | Mejor suma de genes: 168/180 | Mejor ID: 9999440
Generación 99995 | Mejor suma de genes: 170/180 | Mejor ID: 9999538
Generación 99996 | Mejor suma de genes: 167/180 | Mejor ID: 9999632
Generación 99997 | Mejor suma de genes: 167/180 | Mejor ID: 9999790
Generación 99998 | Mejor suma de genes: 164/180 | Mejor ID: 9999822
Generación 99999 | Mejor suma de genes: 165/180 | Mejor ID: 9999902
Generación 100000 | Mejor suma de genes: 167/180 | Mejor ID: 10000070
No se encontró un humano perfecto tras 100000 generaciones.
```

Figure 9. valor p_mejora de 0.80 y p_empeora de 0.10

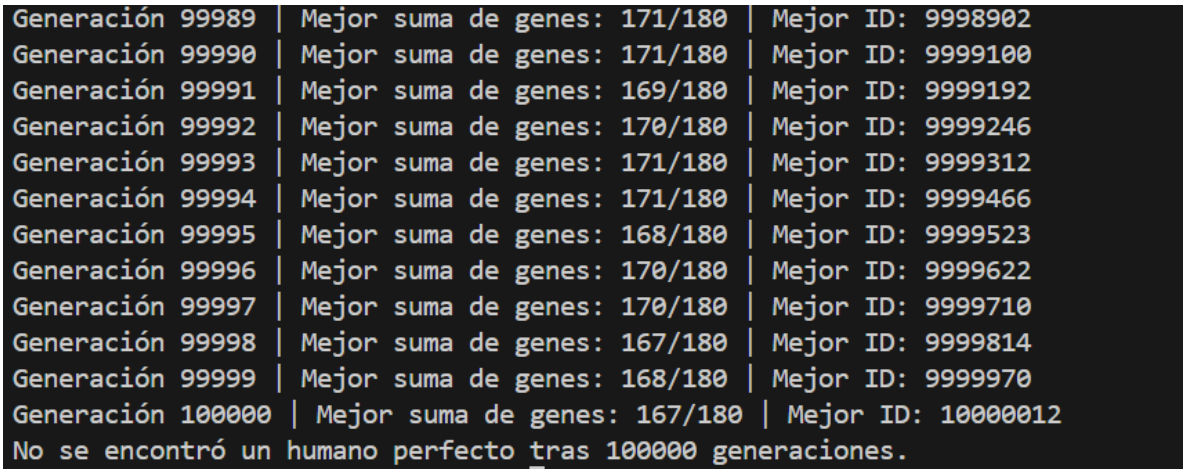

Figure 10. valor p_mejora de 0.80 y p_empeora de 0.10

Se puede apreciar que en ninguna de las 3 corridas en 100000 generaciones se pudo encontrar a un humano perfecto, debido a la naturaleza cambiante del problema y las distintas probabilidades de mejoría o empeoramiento de genes, demostrando que se requiere tener una delimitación un tanto precisa de los parámetros en los genes de cada individuo para lograr encontrar al menos a un humano perfecto.

Explicación de código

Estructuras y parámetros globales

Se fijan los límites GEN_MIN=1, GEN_MAX=9, el tamaño del genoma NUM_GENES=20 y el rango del porcentaje de mutación por individuo [0.10, 0.90]. La clase Individuo (dataclass) encapsula id, genes, porcentaje_mutacion y padres (tupla (id_padre_a, id_padre_b) en los hijos). Además, se tipifican alias para PoblacionTotal (diccionario global por ID) y Generaciones (lista de listas de IDs por generación).

CrearPoblacionInicial(n)
Genera n individuos con genes aleatorios en [1, 9] y un porcentaje_mutacion uniforme en [0.10, 0.90]. Los padres quedan en None porque son fundadores.

FormarParejas(poblacion)
Baraja la población y agrupa de 2 en 2. Si el tamaño no es par lanza ValueError. Esta función impone emparejamiento aleatorio sin sesgo por fitness.

SonHermanos/ElegirReemplazoNoHermano
SonHermanos compara la tupla de padres. ElegirReemplazoNoHermano intenta seleccionar un individuo al azar que no sea el mismo que el “fijo” y que **no** comparta padres, con hasta 10 000 reintentos para robustez.

CrearHijos(padre_a, padre_b, poblacion, id_inicio)

- Si padre_a y padre_b son el mismo individuo, reemplaza al segundo por un no-hermano.
- Si son hermanos, reemplaza al azar a uno para evitar consanguinidad.
- Calcula los **promedios** [(g1+g2)/2] por locus y crea dos hijos: floor (genes abajo) y ceil (genes arriba).
- Asigna padres=(id_a, id_b) y **nuevo porcentaje de mutación** aleatorio a cada hijo. Devuelve la lista [hijo_abajo, hijo_arriba] y los padres efectivos usados.

EsHumanoPerfecto / HayHumanoPerfecto
Detección de perfección: todos los genes = GEN_MAX. La segunda función recorre una población y retorna (True, individuo) si encuentra alguno.

AplicarMutacionUnaVez(individuo, p_mejora=0.89, p_empeora=0.1)

- Primero decide si el individuo muta comparando random() contra su porcentaje_mutacion.
- Si muta, elige **un índice** al azar y modifica el alelo respetando las probabilidades: mejora, empeora o cambio aleatorio distinto. Garantiza mantener el alelo dentro de [GEN_MIN, GEN_MAX] y además evita que el "cambio aleatorio" deje el mismo valor.

CrearHijosParaParejas(parejas, poblacion, siguiente_id, aplicar_mutacion=True)
Itera sobre las parejas y llama CrearHijos. Si aplicar_mutacion es True, intenta mutar a cada hijo una vez. Incrementa siguiente_id en 2 por pareja. Devuelve la lista consolidada de nuevos hijos y el próximo ID libre.

InicializarEstructuras / ActualizarEstructurasTrasGeneracion
El inicializador construye:

- poblacion_total: diccionario id → Individuo para historial global.
- generaciones: lista con la **Generación 0** (IDs de fundadores).
- siguiente_id: el siguiente entero disponible.

El actualizador agrega los **nuevos hijos** al índice global, añade sus IDs como la nueva generación y verifica si apareció un individuo perfecto dentro de estos hijos.

EjecutarGeneracion
Engloba los pasos de una generación: formar parejas, crear hijos (con o sin mutación),

registrar estructuras y comprobar perfección. Retorna la población resultante (los hijos), el siguiente_id actualizado y el indicador/perfecto si se alcanzó la meta.

SumaTotalGenes / MejorDePoblacion
Define un criterio de "fitness" simple: la suma de los 20 genes. Se usa para **reportar** el mejor individuo de cada generación (no para seleccionar padres).

SimularHastaPerfecto(...)
Controla el ciclo completo:

- Fija semilla si se provee, para reproducibilidad.
- Inicializa estructuras y muestra el **mejor** de la Generación 0.
- Revisa si ya hay un perfecto en la población inicial.
- Itera generaciones hasta max_generaciones. En cada paso, ejecuta una generación, imprime progreso opcionalmente y detiene si aparece un perfecto.
- Devuelve un *tuple* con: si se encontró perfecto, el individuo perfecto (si aplica), las generaciones registradas, el índice global, cuántas generaciones se procesaron y el siguiente ID.

Bloque de ejecución
Crea poblacion0 = CrearPoblacionInicial(100) y llama SimularHastaPerfecto con max_generaciones=100000, aplicar_mutacion=True, semilla=42 e impresión de progreso activada.

Anexos

Conjunto de valores iniciales para definir a la primera población, el número máximo de generaciones, decidir si aplicar mutaciones, definir la semilla para los valores aleatorios y para decidir si imprimir el progreso de las generaciones o no:

```
poblacion0 = CrearPoblacionInicial(100)
encontro, perfecto, generaciones, poblacion_total, gens_proc, siguiente_id = SimularHastaPerfecto(
    poblacion_inicial=poblacion0,
    max_generaciones=100000,
    aplicar_mutacion=True,
    semilla=42,
    imprimir_progreso=True
)
```

Conclusiones

La simulación reproduce de forma controlada la dinámica básica de un algoritmo evolutivo: inicialización aleatoria, reproducción por cruce, variación por mutación y evaluación mediante un criterio de aptitud. Aunque el apareamiento es aleatorio (no hay selección por fitness para elegir padres), el uso de promedios gen-a-gen más mutación con fuerte sesgo a mejorar favorece la aparición progresiva de individuos con mayor suma de genes. El criterio de paro basado en un individuo con todos los genes en 9 es exigente y permite observar cómo influyen la tasa de mutación por individuo, la regla de mutación y la semilla aleatoria en el tiempo de convergencia. En conjunto, la práctica muestra cómo decisiones de diseño aparentemente simples (cómo cruzar, cuándo mutar y cómo medir la aptitud) determinan el comportamiento global del sistema y la probabilidad de alcanzar soluciones óptimas.