Universidad de Buenos Aires
Facultad de Ingeniería

Teoría de Algoritmos 75.29/95.06/TB024
Trabajo Práctico 3 - Curso Echevarría
1er Cuatrimestre 2025

Grupo 15
Integrantes: Bautista Corti, Tomas Amundarain, Andres Moyano, Leandro Brizuela

Índice

PROBLEMA 1

Instancia de A y B

Sea A, un conjunto de 5 enteros positivos:

$$A = [1, 1, 1, 1, 10]$$

Sea B, un número entero positivo:

$$B = 10$$

Sea S, la solución dada por el algoritmo proporcionado en el enunciado:

$$S = [1, 1, 1, 1]$$

Y sea S' la solución óptima para el conjunto A y el número objetivo B:

$$S' = [10]$$

Se pide que para la instancia de A y B la solución S sea:

$$\sum_{1}^{n} s_i < \frac{1}{2} \sum_{1}^{m} s'_j = 4 < 5$$

El la suma del conjunto S' es la mayor posible y es superior a la encontrada por el algoritmo de estrategia greedy del enunciado, se cumple además la condición pedida para la instancia.

Supuestos

Para este problema se tomaron los siguientes supuestos:

- El algoritmo soporta un conjunto A y un número B enteros y positivos. No se considera la posibilidad de números negativos o valores no numéricos.
- El óptimo S_{opt} no puede ser mayor que B. Los conjuntos A y B deben ser resolubles, no se admiten situaciones como: A = [10000, 10005, 5600], B = 5

Diseño

Para resolver el problema, se decidió realizar una pequeña modificación al algoritmo original; ahora se ordenan en orden descendente los números del conjunto A antes de iniciar con la regla greedy del algoritmo anterior. Al realizar este cambio se sigue garantizando que $\sum S \leq B$ pero garantiza también la calidad de aproximación solicitada en el enunciado.

Esta decisión se apoya en la literatura clásica del problema de la mochila[1], donde este tipo de heurística (basada en agregar elementos en orden descendente y hasta que no quepa más) es ampliamente estudiada. Según Martello y Toth (1990), el algoritmo greedy clásico tiene un rendimiento de tiempo O(nlogn) por el ordenamiento inicial y O(n) para construir la

[1] El problema del subconjunto factible puede verse como una variante del **Subset Sum**, que a su vez es una versión simplificada del problema de la mochila (Knapsack Problem), en la que los pesos y beneficios de los elementos coinciden. Por lo tanto, los resultados y garantías del algoritmo greedy para la mochila se aplican directamente en este contexto.

solución. Además, cuando se compara con el ítem de mayor beneficio individual, el algoritmo ofrece una garantía de rendimiento de al menos $\frac{1}{2}$ del valor óptimo:

> "A more effective algorithm is to consider them according to increasing indices [after sorting], and insert each new item into the knapsack if it fits [...] This is the most popular heuristic approach to KP [...] The worst-case performance ratio is ½ [...] the time complexity is O(n), plus O(nlogn) for the initial sorting." (Martello & Toth, 1990, p. 28)

En nuestro caso, como el algoritmo intenta maximizar la suma de S, o lo que es equivalente, minimizar la diferencia con respecto a B, esto garantiza que se devuelva una solución cercana a B (ya sea subóptima pero cercana, o el mejor valor factible $m = max\{a_i \in A: a_i \leq B\}$). Por lo tanto, esta variante greedy cumple con la condición de aproximación mínima exigida.

Pseudocódigo

```
subconjunto_factible(A, B)
    S={}
    T=0
    Ordenar A de forma descendente
    Para i=1,2,...,n
        Si T + A[i] <= B, entonces:
            S = S ∪ A[i]
            T = T + A[i]
        fin si
    fin para
    devolver S
```

Para almacenar los valores que se van acumulando en S se uso una lista predeterminada de Python.

Seguimiento

$A\ (ordenado)\ =\ [9, 8, 7, 5, 2]$, B = 10, $S_{opt}\ =\ [8,\ 2]$

1) Como $0\ +\ 9\ \leq\ 10$ se agrega 9 al S_{greedy}
2) Como $9\ +\ 8\ \geq\ 10$ no se agrega 9 al S_{greedy}
3) Como $9\ +\ 7\ \geq\ 10$ no se agrega 7 al S_{greedy}
4) Como $9\ +\ 5\ \geq\ 10$ no se agrega 5 al S_{greedy}
5) Como $9\ +\ 2\ \geq\ 10$ no se agrega 2 al S_{greedy}
6) Se finaliza con un S_{greedy} con [9]

$$\sum_{1}^{n} s_i \geq \frac{1}{2} \sum_{1}^{m} s_{opt\ j} \Rightarrow 9 \geq 5$$

Complejidad

Se asume que el ordenamiento de las postas en la carrera tiene complejidad de $O(nlog(n))$ dado a que Python utiliza el algoritmo de ordenamiento TimSort que en promedio cuenta con esa complejidad.

A cada elemento de A se lo consulta una vez con una complejidad de $O(1)$ por cada una de estas operaciones. Dado a que se realizan n de estas consultas la complejidad dentro del for es de $O(n)$.

La complejidad temporal resultante es una suma de estos procesos:

$$O(n + nlog(n)) \simeq O(nlog(n))$$

Informe

El algoritmo planteado resulta ser una buena aproximación para el problema de subconjunto factible, un problema derivado del subset sum y siendo también NP Hard.
Se obtienen resultados mejores a los que se obtendrían con la aproximación mínima planteada en el enunciado ($\sum_{1}^{n} s_i \geq \frac{1}{2} \sum_{1}^{m} s_{opt\ j}$). A continuación se mostrará 5 ejecuciones del algoritmo; se muestra los resultados obtenidos con un A de tamaño[2] 1000. Se incluyen el B, la S obtenida, la óptima y sus comparaciones, los tiempos de ambos algoritmos redondeados a 6 cifras significativas y la semilla aleatoria usada:

Ejecución 1:

B: 2666
S_{greedy}: [2665]
S_{opt}: [1800, 866]
$2665 \geq 1333$
Tiempo aproximación: 0.00155830
Tiempo del óptimo: 3.50477
Seed: 527349

Ejecución 2:

B: 4109

[2] Se muestra solo el tamaño porque algunos A contienen muchos elementos. De cualquier forma el experimento se puede replicar con la semilla aleatoria dada.

S_{greedy}: [4105, 3]
S_{opt}: [954, 2460, 3, 692]
$4108 \geq 2054$
Tiempo aproximación: 0.00169960
Tiempo del óptimo: 3.46091
Seed: 952304

Ejecución 3:

B: 2597
S_{greedy}: [2594]
S_{opt}: [602, 392, 659, 794, 150]
$2594 \geq 1298$
Tiempo aproximación: 0.000730200
Tiempo del óptimo: 2.21115
Seed: 231546

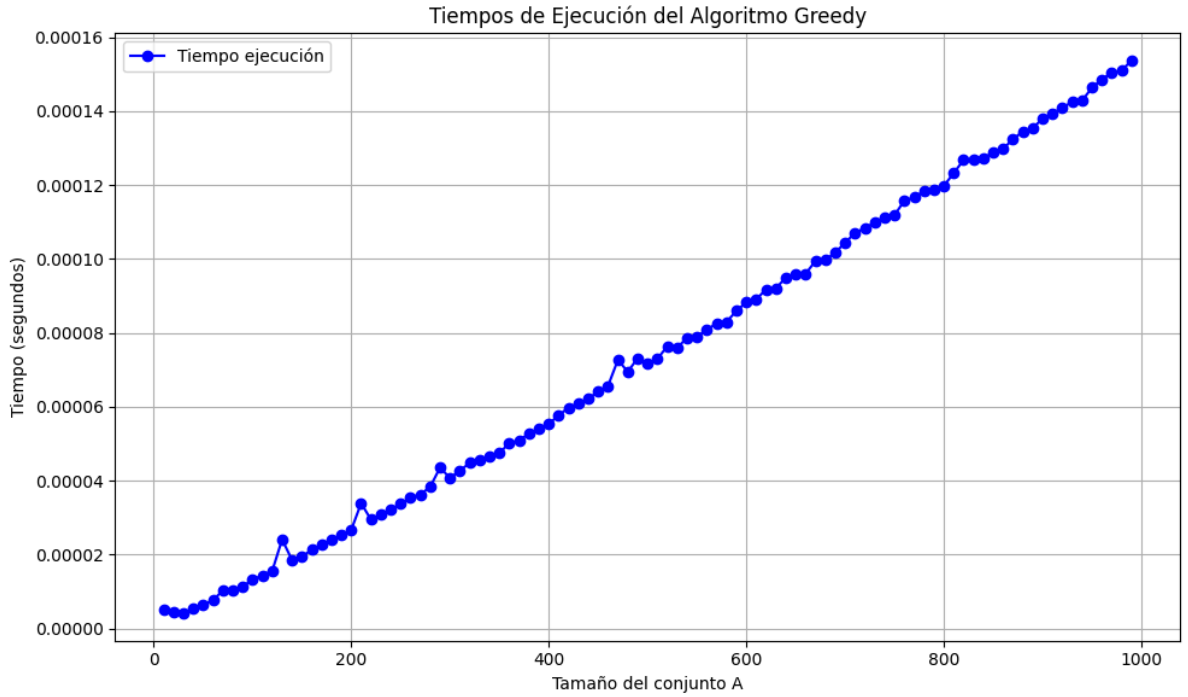
Ejecución 4:

B: 2542
S_{greedy}: [2540, 1]
S_{opt}: [26, 283, 958, 644, 631]
$2541 \geq 1271$
Tiempo aproximación: 0.000680100
Tiempo del óptimo: 2.04720
Seed: 193213

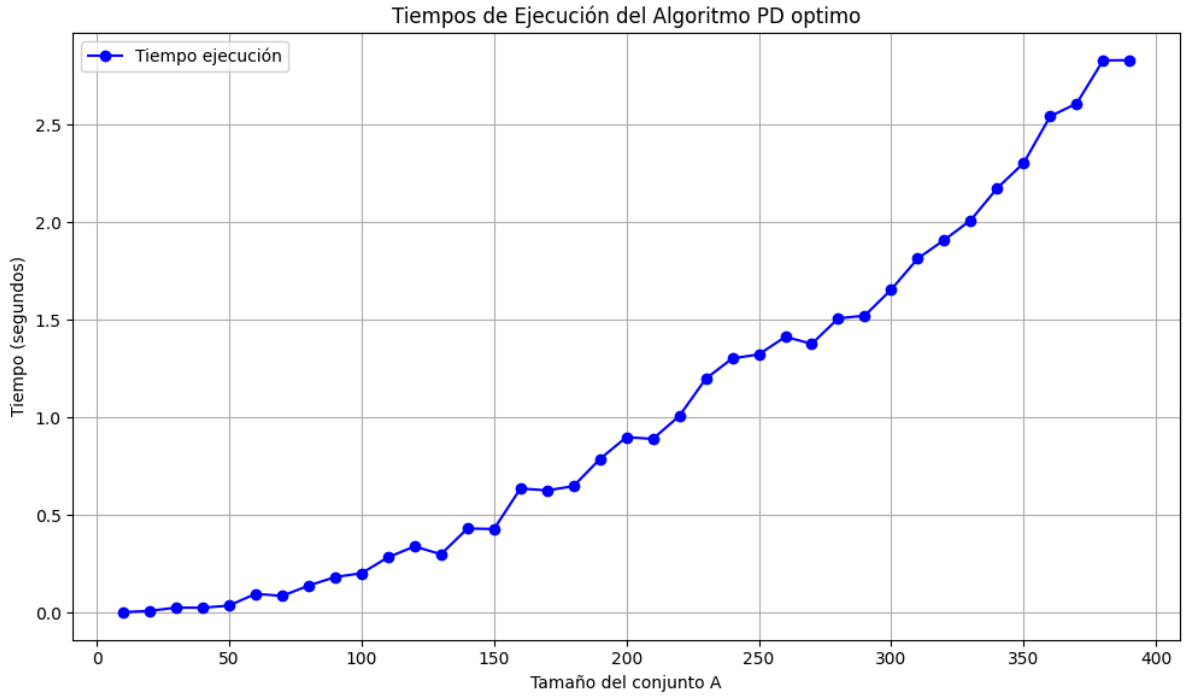
Ejecución 5:

B: 1580
S_{greedy}: [1580]
S_{opt}: [597, 983]
$1580 \geq 790$
Tiempo aproximación: 0.00109110
Tiempo del óptimo: 1.15702
Seed: 956048

Para graficar los tiempos de ejecución del algoritmo planteado prueban A con 98 tamaños diferentes desde 10 hasta 990 con saltos de 10 en 10. Los números anteriores serán los mismos para no afectar los resultados con cantidades mayores de números en A (si se empieza con [1, 57, 9, 2, 453, 30, 8, 16, 79, 43] los siguientes 10 números del próximo salto serán 10 nuevos aleatorios que se consolidaran para el salto posterior y así sucesivamente). El valor de los números ronda entre 1 y 10000 y el B es el mayor número del A actual garantizando siempre resolubilidad y un B razonable.

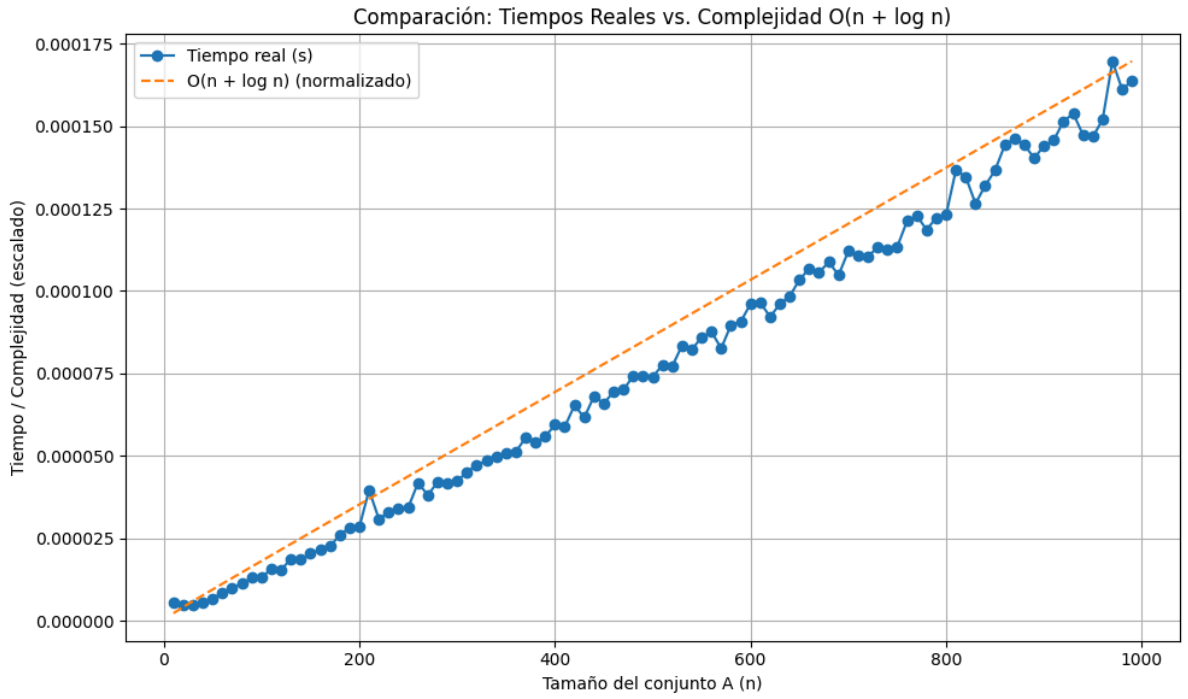


Adicionalmente se graficará el algoritmo que siempre encuentra el S óptimo que es mucho más lento, consecuentemente se debió graficar un set de datos menor y más restrictivo para un gráfico con menor ruido y más entendible.
Se prueban A con 38 tamaños diferentes desde 10 hasta 390 con saltos de 10 en 10. Los números anteriores serán los mismos para no afectar los resultados con cantidades mayores de números en A. El valor de los números ronda entre 1 y 500 y B es un 40% de la sumatoria del A actual garantizando siempre resolubilidad y un B razonable.



Como se ve en el gráfico para un set de datos menor se ve que la curva tenderá al alza a medida que el tamaño de A aumenta. Esto corresponde a un algoritmo exponencial, lo cual es correcto dado a que para la comparación se usó un algoritmo que utiliza la programación dinámica para la resolución de este problema NP hard.

El algoritmo planteado no se aleja mucho de la complejidad esperada:
Se prueban A con 98 tamaños diferentes desde 10 hasta 990 con saltos de 10 en 10. Los números anteriores serán los mismos para no afectar los resultados con cantidades mayores de números en A. El valor de los números ronda entre 1 y 10000 y el B es un 40% de la sumatoria del A actual garantizando siempre resolubilidad y un B razonable.



Se concluye que el algoritmo es una buena aproximación que aunque es más lento que el algoritmo original, sigue siendo mucho más rápido que el algoritmo con estrategia PD que encuentra el óptimo siempre.

PROBLEMA 2

Problema de la Secretaria

Para poder resolver el problema vamos a recurrir a otro algoritmo aplicado a un problema similar.

“Supongamos que tenemos que elegir secretaria de entre 100 candidatas. Solo podemos entrevistar a cada persona una vez. Después de cada entrevista, hay que decidir si la contratamos o no. Si decidimos que no, perdemos la oportunidad para siempre. Si entrevistamos a 99 sin haber elegido, tenemos que contratar a la número 100.”

Es un problema muy similar al nuestro y se lo conoce también como el problema del matrimonio óptimo, el problema de selección del mejor candidato, o como una regla de parada óptima.

La resolución del problema consiste en aplicar la regla del 1/e con $e \approx 2.781$. Hay que rechazar los primeros r = n/e candidatos, solo los entrevistamos sin contratar a nadie. A partir del siguiente al r-avo candidato, elegir al primero que sea mejor que todos los anteriores.

De esta forma se garantiza que la probabilidad de elegir al mejor candidato es al menos 1/e aproximadamente 0.368, independientemente de la cantidad de postulantes.

¿Cómo se aplica a nuestro problema?

Un vendedor recibe una secuencia aleatoria de ofertas, similar a los candidatos aleatorios. Puede ver el valor de cada oferta a medida que llega, con los candidatos funciona igual, tengo que entrevistarlo para conocerlo. En ambos caso tengo que decidir en el momento si acepto o no la oferta, o si lo contrato o no. Una vez descartada una oferta o candidato, no puedo volver atrás. El objetivo es aceptar la mejor oferta o contratar el mejor candidato.

Ambos problemas comparten la misma estructura probabilística: una secuencia de decisiones irreversibles sobre ítems vistos en orden aleatorio, con el objetivo de maximizar la probabilidad de elegir el mejor. Y garantiza una probabilidad de éxito mayor al 25% (al menos 36.8%).

Supuestos

- Suponemos que las ofertas llegan en orden aleatorio.
- El vendedor conoce la cantidad de ofertas totales de antemano.
- No se puede volver a una oferta pasada.
- Todas las ofertas son distintas (para evitar desempates).
- Las decisiones del vendedor se basan solo en las ofertas vistas hasta el momento y no en el valor real de la mejor oferta.

Diseño

Garantía

El algoritmo de la secretaria ya fue probado en muchos contextos como óptimo o cuasi óptimo.
Véase Ferguson (1989) y Dynkin (1963) para una descripción formal del modelo y su análisis.

En particular, el valor de corte r = n/e es demostrado matemáticamente que maximiza la probabilidad de éxito. En el peor de los casos (pocas ofertas/candidatos), aún eligiendo un valor más chico r = n/4, se garantiza una probabilidad razonable mayor al 25%.

Pseudocódigo

```
def elegir_mejor_oferta(ofertas):
      n = len(ofertas)
      r = int(n / math.e)
      mejor_hasta_ahora = max(ofertas[:r])

      for i in range(r, n):
            if ofertas[i] > mejor_hasta_ahora:
                  return ofertas[i]

      return ofertas[-1]   # si nunca apareció una mejor, aceptar
la última.
```

Estructura de Datos

Lista de enteros o floats y una variable para seguir el máximo visto.

Seguimiento

ofertas = [7, 3, 9, 5, 2, 8, 6, 10, 1, 4] # n=10
r = 3 (10/e ≈ 3.68)

Fase de observación: [7, 3, 9] → mejor_hasta_ahora = 9

Luego:
- 5 → no mayor
- 2 → no
- 8 → no
- 6 → no
- 10 → mayor → seleccionar 10

Complejidad

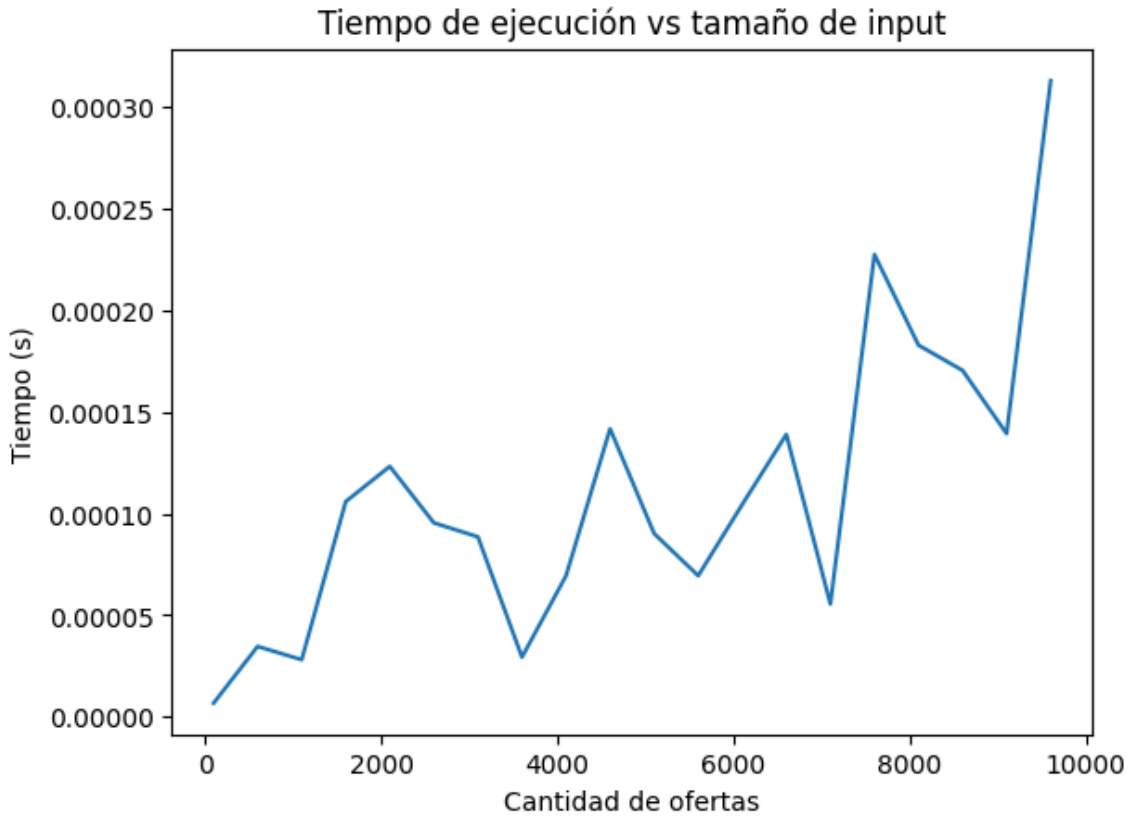
O(n), se recorre la lista una sola vez.

Informe

El algoritmo implementado sigue la estrategia del problema del secretario, donde se descartan las primeras n/e ofertas y luego se acepta la primera que supere a todas las anteriores. Esta estrategia garantiza una probabilidad mayor a 1/4 de elegir la mejor oferta, sin importar cuántas haya.

Para verificar su rendimiento, medimos el tiempo de ejecución en función de la cantidad de ofertas. Se usaron listas de ofertas aleatorias con tamaños entre 100 y 9500. Los tiempos se midieron usando time.perf_counter() y se graficaron.

Como se ve en el gráfico, el tiempo de ejecución en general crece con el tamaño de entrada, aunque no de forma perfectamente lineal. Esto se debe a que los tiempos son muy pequeños y pueden verse afectados por el entorno (otros procesos, precisión del reloj, etc.). A pesar de eso, los resultados son coherentes con lo esperado para un algoritmo de complejidad lineal O(n), ya que en ningún caso el tiempo crece abruptamente y sigue una tendencia razonable.



PROBLEMA 3

Definición Formal de los lenguajes simples:

A continuación se definirá formalmente un autómata finito determinístico para cada uno de los lenguajes simples:

1) **Lenguaje 1:** $\{w \,/\, w\ (comienza\ con\ "1")\}$

   Sea $M_1 = (Q1\ ,\ \sum\ , T1\ ,\ q1_0,\ F1)$ con:

   - Conjunto de estados: $Q1 = \{q1_0,\ q1_1,\ q1_{err}\}$
   - Alfabeto: $\sum = \{0, 1\}$
   - Función de transición $T1$:

-	0	1
$q1_0$	$q1_{err}$	$q1_1$
$q1_1$	$q1_1$	$q1_1$
$q1_{err}$	$q1_{err}$	$q1_{err}$

   - Estado inicial: $q1_0$
   - Conjunto de estados de aceptación: $F1 = \{q1_1\}$

2) **Lenguaje 2:** $\{w \,/\, w\ (tiene\ cuando\ mucho\ un\ "0")\}$

   Sea $M_2 = (Q2\ ,\ \sum\ , T2\ ,\ q2_0,\ F2)$ con:

   - Conjunto de estados: $Q2 = \{q2_0,\ q2_1,\ q2_{arr}\}$
   - Alfabeto: $\sum = \{0, 1\}$
   - Función de transición $T2$:

-	0	1
$q2_0$	$q2_1$	$q2_0$
$q2_1$	$q2_{err}$	$q2_1$
$q2_{err}$	$q2_{err}$	$q2_{err}$

- Estado inicial: $q2_0$
- Conjunto de estados de aceptación: $F2 = \{q2_0, q2_1\}$

3) Lenguaje 3: $\{w / w\ (tiene\ un\ número\ par\ de\ "1")\}$

Sea $M_3 = (Q3\ ,\ \sum\ , T3\ ,\ q3_0, F3)$ con:

- Conjunto de estados: $Q3 = \{q3_0, q3_1\}$
- Alfabeto: $\sum = \{0, 1\}$
- Función de transición $T3$:

-	0	1
$q3_0$	$q3_0$	$q3_1$
$q3_1$	$q3_1$	$q3_0$

- Estado inicial: $q3_0$
- Conjunto de estados de aceptación: $F3 = \{q3_0\}$

■ **Nota:** Inicialmente empezamos con un número par de “1” ya que es 0**.**
  - $q3_0$ representa que hay una cantidad par de unos.
  - $q3_1$ representa que hay una cantidad impar de unos.

4) Lenguaje 4: $\{w / w\ (tiene\ uno\ o\ dos\ "0")\}$

Sea $M_4 = (Q4\ ,\ \sum\ , T4\ ,\ q4_0, F4)$ con:

- Conjunto de estados: $Q4 = \{q4_0, q4_1, q4_2, q4_{err}\}$
- Alfabeto: $\sum = \{0, 1\}$
- Función de transición $T4$:

-	0	1
$q4_0$	$q4_1$	$q4_0$
$q4_1$	$q4_2$	$q4_1$
$q4_2$	$q4_{err}$	$q4_2$
$q4_{err}$	$q4_{err}$	$q4_{err}$

- Estado inicial: $q4_0$
- Conjunto de estados de aceptación: $F4 = \{q4_1, q4_2\}$

Definición Formal de los lenguajes A y B:

A continuación, se definirá formalmente un autómata finito determinístico para cada uno de los lenguajes A y B:

1) Lenguaje A:

Sea $M_A = (QA, \sum, TA, qA_0, FA)$ con:

- Conjunto de estados: $QA = \{qA_0, qA_1, qA_2, qA_{err}\}$
- Alfabeto: $\sum = \{0, 1\}$
- Función de transición TA :

-	0	1
qA_0	qA_{err}	qA_1
qA_1	qA_2	qA_1
qA_2	qA_{err}	qA_2
qA_{err}	qA_{err}	qA_{err}

- Estado Inicial: qA_0
- Conjunto de estados de aceptación: $FA = \{qA_1, qA_2\}$

2) Lenguaje B:

Sea $M_B = (QB, \sum, TB, qB_0, FB)$ con:

- Conjunto de estados:
  $QB = \{qB_{par_0}, qB_{par_1}, qB_{par_2}, qB_{impar_0}, qB_{impar_1}, qB_{impar_2}, qB_{err}\}$
- Alfabeto: $\sum = \{0, 1\}$
- Función de transición TB :

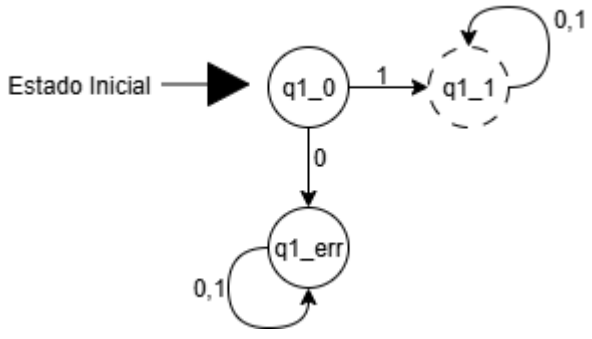
-	0	1
qB_{par_0}	qB_{par_1}	qB_{impar_0}
qB_{par_1}	qB_{par_2}	qB_{impar_1}
qB_{par_2}	qB_{err}	qB_{impar_2}
qB_{impar_0}	qB_{impar_1}	qB_{par_0}
qB_{impar_1}	qB_{impar_2}	qB_{par_1}
qB_{impar_2}	qB_{err}	qB_{par_2}
qB_{err}	qB_{err}	qB_{err}

- Estado inicial: qB_{par_0}
  - **Nota:** Tomamos que la cantidad 0 de “1” es par.
- Conjunto de estados de aceptación: $FB = \{qB_{par_1}, qB_{par_2}\}$

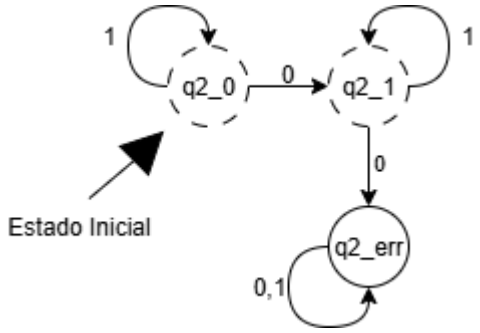
Representación gráfica de los Autómatas

A continuación se mostrará gráficamente los autómatas definidos anteriormente:

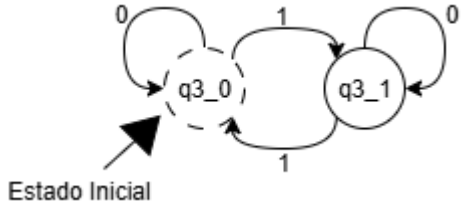
1) **Autómata M_1:**



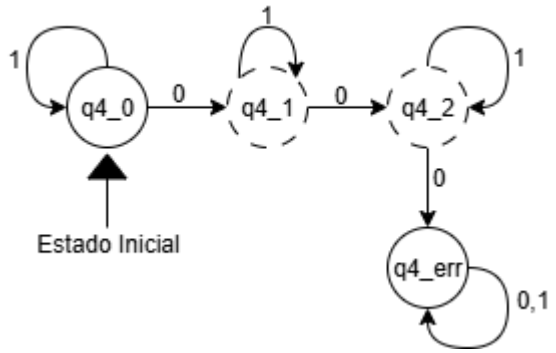
2) Autómata M_2:



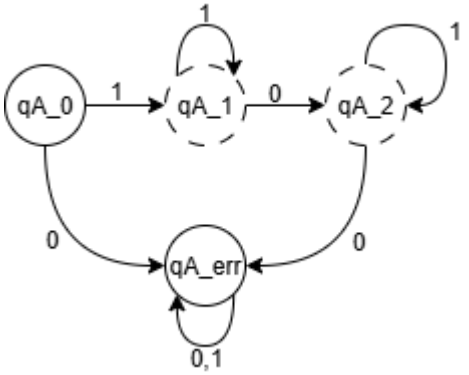
3) Autómata M_3:



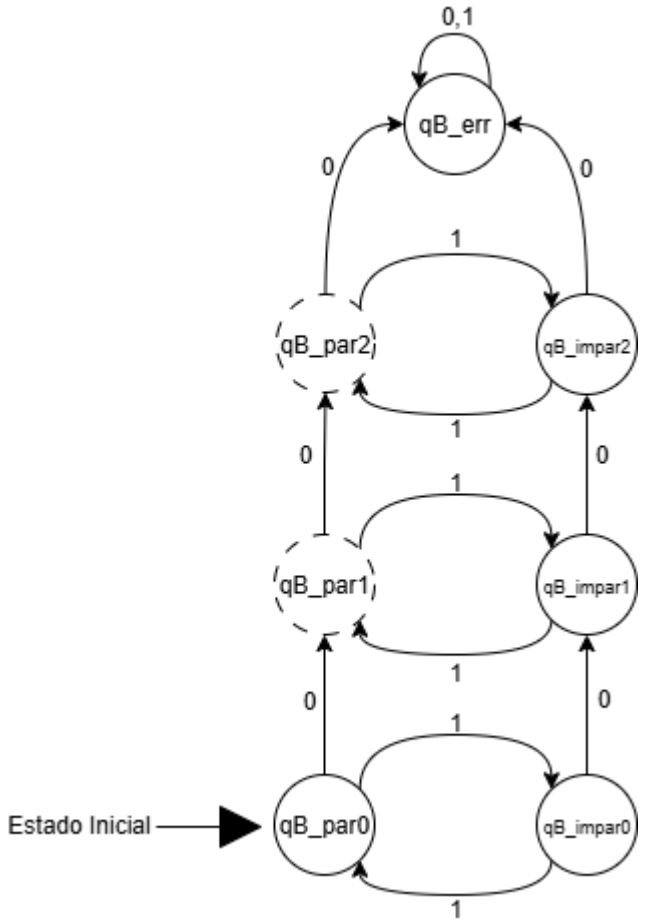
4) Autómata M_4:



5) Autómata M_A:



6) **Autómata M_B:**



Ejemplos

A continuación se mostrarán ejemplos de aceptación y rechazo de una cadena para cada autómata:

1) **Ejemplos para M_1:**

- **Ejemplo de aceptación:** Supongamos que recibimos la cadena 101011

1. Estamos en el estado inicial $q1_0$ y recibimos un 1 → Pasamos al estado $q1_1$
2. Estamos en el estado $q1_1$ y recibimos un 0 → Nos mantenemos en el estado $q1_1$
3. Estamos en el estado $q1_1$ y recibimos un 1 → Nos mantenemos en el estado $q1_1$
4. Estamos en el estado $q1_1$ y recibimos un 0 → Nos mantenemos en el estado $q1_1$
5. Estamos en el estado $q1_1$ y recibimos un 1 → Nos mantenemos en el estado $q1_1$
6. Estamos en el estado $q1_1$ y recibimos un 1 → Nos mantenemos en el estado $q1_1$

- Terminamos de procesar la cadena y quedamos en el estado $q1_1$ (el cual es estado de aceptación) → La cadena 101011 es aceptada por el autómata M_1.

- **Ejemplo de rechazo:** Supongamos que recibimos la cadena 010100

1. Estamos en el estado inicial $q1_0$ y recibimos un 0 → Pasamos al estado $q1_{err}$
2. Estamos en el estado $q1_{err}$ y recibimos un 1 → Nos mantenemos en el estado $q1_{err}$
3. Estamos en el estado $q1_{err}$ y recibimos un 0 → Nos mantenemos en el estado $q1_{err}$
4. Estamos en el estado $q1_{err}$ y recibimos un 1 → Nos mantenemos en el estado $q1_{err}$
5. Estamos en el estado $q1_{err}$ y recibimos un 0 → Nos mantenemos en el estado $q1_{err}$
6. Estamos en el estado $q1_{err}$ y recibimos un 0 → Nos mantenemos en el estado $q1_{err}$

- Terminamos de procesar la cadena y quedamos en el estado $q1_{err}$ (el cual no es estado de aceptación) → La cadena 010100 es rechazada por el autómata M_1.

2) Ejemplos para M_2:

- **Ejemplo de aceptación:** Supongamos que recibimos la cadena 101111

1. Estamos en el estado inicial $q2_0$ y recibimos un 1 → Nos mantenemos en el estado inicial $q2_0$
2. Estamos en el estado $q2_0$ y recibimos un 0 → Pasamos al estado $q2_1$
3. Estamos en el estado $q2_1$ y recibimos un 1 → Nos mantenemos en el estado $q2_1$
4. Estamos en el estado $q2_1$ y recibimos un 1 → Nos mantenemos en el estado $q2_1$
5. Estamos en el estado $q2_1$ y recibimos un 1 → Nos mantenemos en el estado $q2_1$
6. Estamos en el estado $q2_1$ y recibimos un 1 → Nos mantenemos en el estado $q2_1$

- Terminamos de procesar la cadena y quedamos en el estado $q2_1$ (el cual es estado de aceptación) → La cadena 101111 es aceptada por el autómata M_2.

- **Ejemplo de rechazo:** Supongamos que recibimos la cadena 010100

1. Estamos en el estado inicial $q2_0$ y recibimos un 0 → Pasamos al estado $q2_1$
2. Estamos en el estado $q2_1$ y recibimos un 1 → Nos mantenemos en el estado $q2_1$
3. Estamos en el estado $q2_1$ y recibimos un 0 → Pasamos al estado $q2_{err}$
4. Estamos en el estado $q2_{err}$ y recibimos un 1 → Nos mantenemos en el estado $q2_{err}$
5. Estamos en el estado $q2_{err}$ y recibimos un 0 → Nos mantenemos en el estado $q2_{err}$
6. Estamos en el estado $q2_{err}$ y recibimos un 0 → Nos mantenemos en el estado $q2_{err}$

- Terminamos de procesar la cadena y quedamos en el estado $q2_{err}$ (el cual no es estado de aceptación) → La cadena 010100 es rechazada por el autómata M_2.

3) Ejemplos para M_3:

- **Ejemplo de aceptación:** Supongamos que recibimos la cadena 010100

1. Estamos en el estado inicial $q3_0$ y recibimos un 0 → Nos mantenemos en el estado inicial $q3_0$.
2. Estamos en el estado $q3_0$ y recibimos un 1 → Pasamos al estado $q3_1$
3. Estamos en el estado $q3_1$ y recibimos un 0 → Nos mantendremos en el estado $q3_1$
4. Estamos en el estado $q3_1$ y recibimos un 1 → Volvemos al estado $q3_0$
5. Estamos en el estado $q3_0$ y recibimos un 0 → Nos mantenemos en el estado $q3_0$
6. Estamos en el estado $q3_0$ y recibimos un 0 → Nos mantenemos en el estado $q3_0$

- Terminamos de procesar la cadena y quedamos en el estado $q3_0$ (el cual es estado de aceptación) → La cadena 010100 es aceptada por el autómata M_3.

- **Ejemplo de rechazo:** Supongamos que recibimos la cadena 101111

1. Estamos en el estado inicial $q3_0$ y recibimos un 1 → Pasamos al estado $q3_1$
2. Estamos en el estado $q3_1$ y recibimos un 0 → Nos mantenemos en el estado $q3_1$
3. Estamos en el estado $q3_1$ y recibimos un 1 → Volvemos al estado $q3_0$
4. Estamos en el estado $q3_0$ y recibimos un 1 → Volvemos al estado $q3_1$
5. Estamos en el estado $q3_1$ y recibimos un 1 → Volvemos al estado $q3_0$
6. Estamos en el estado $q3_0$ y recibimos un 1 → Volvemos al estado $q3_1$

- Terminamos de procesar la cadena y quedamos en el estado $q3_1$(el cual no es estado de aceptación) → La cadena 101111 es rechazada por el autómata M_3.

4) Ejemplos para M_4:

- **Ejemplo de aceptación:** Supongamos que recibimos la cadena 101111

1. Estamos en el estado inicial $q4_0$ y recibimos un 1 → Nos mantenemos en el estado inicial $q4_0$.
2. Estamos en el estado $q4_0$ y recibimos un 0 → Pasamos al estado $q4_1$
3. Estamos en el estado $q4_1$ y recibimos un 1 → Nos mantenemos en el estado $q4_1$
4. Estamos en el estado $q4_1$ y recibimos un 1 → Nos mantenemos en el estado $q4_1$
5. Estamos en el estado $q4_1$ y recibimos un 1 → Nos mantenemos en el estado $q4_1$
6. Estamos en el estado $q4_1$ y recibimos un 1 → Nos mantenemos en el estado $q4_1$

- Terminamos de procesar la cadena y quedamos en el estado $q4_1$ (el cual es estado de aceptación) → La cadena 101111 es aceptada por el autómata M_4.

- **Ejemplo de rechazo:** Supongamos que recibimos la cadena 010100

1. Estamos en el estado inicial $q4_0$ y recibimos un 0 → Pasamos al estado $q4_1$
2. Estamos en el estado $q4_1$ y recibimos un 1 → Nos mantenemos en el estado $q4_1$
3. Estamos en el estado $q4_1$ y recibimos un 0 → Pasamos al estado $q4_2$
4. Estamos en el estado $q4_2$ y recibimos un 1 → Nos mantenemos en el estado $q4_2$
5. Estamos en el estado $q4_2$ y recibimos un 0 → Pasamos al estado $q4_{err}$
6. Estamos en el estado $q4_{err}$ y recibimos un 0 → Nos mantenemos en el estado $q4_{err}$

- Terminamos de procesar la cadena y quedamos en el estado $q4_{err}$ (el cual no es estado de aceptación) → La cadena 010100 es rechazada por el autómata M_4.

5) Ejemplos para M_A:

- **Ejemplo de aceptación:** Supongamos que recibimos la cadena 101111

1. Estamos en el estado inicial qA_0 y recibimos un 1 → Pasamos al estado qA_1
2. Estamos en el estado qA_1 y recibimos un 0 → Pasamos al estado qA_2
3. Estamos en el estado qA_2 y recibimos un 1 → Nos mantenemos en el estado qA_2
4. Estamos en el estado qA_2 y recibimos un 1 → Nos mantenemos en el estado qA_2
5. Estamos en el estado qA_2 y recibimos un 1 → Nos mantenemos en el estado qA_2
6. Estamos en el estado qA_2 y recibimos un 1 → Nos mantenemos en el estado qA_2

- Terminamos de procesar la cadena y quedamos en el estado qA_2 (el cual es estado de aceptación) → La cadena 101111 es aceptada por el autómata M_A.

- **Ejemplo de rechazo:** Supongamos que recibimos la cadena 101011

1. Estamos en el estado inicial qA_0 y recibimos un 1 → Pasamos al estado qA_1
2. Estamos en el estado qA_1 y recibimos un 0 → Pasamos al estado qA_2
3. Estamos en el estado qA_2 y recibimos un 1 → Nos mantenemos en el estado qA_2
4. Estamos en el estado qA_2 y recibimos un 0 → Pasamos al estado qA_{err}
5. Estamos en el estado qA_{err} y recibimos un 1 → Nos mantenemos en el estado qA_{err}
6. Estamos en el estado qA_{err} y recibimos un 1 → Nos mantenemos en el estado qA_{err}

- Terminamos de procesar la cadena y quedamos en el estado qA_{err} (el cual no es estado de aceptación) → La cadena 101011 es rechazada por el autómata M_A.

6) Ejemplos para M_B:

- **Ejemplo de aceptación:** Supongamos que recibimos la cadena 101011

1. Estamos en el estado inicial qB_{par_0} y recibimos un 1 → Pasamos al estado qB_{impar_0}
2. Estamos en el estado qB_{impar_0} y recibimos un 0 → Pasamos al estado qB_{impar_1}
3. Estamos en el estado qB_{impar_1} y recibimos un 1 → Pasamos al estado qB_{par_1}
4. Estamos en el estado qB_{par_1} y recibimos un 0 → Pasamos al estado qB_{par_2}
5. Estamos en el estado qB_{par_2} y recibimos un 1 → Pasamos al estado qB_{impar_2}
6. Estamos en el estado qB_{impar_2} y recibimos un 1 → Volvemos al estado qB_{par_2}

- Terminamos de procesar la cadena y quedamos en el estado qB_{par_2} (el cual es estado de aceptación) → La cadena 101011 es aceptada por el autómata M_B.

- **Ejemplo de rechazo:** Supongamos que recibimos la cadena 101111

1. Estamos en el estado inicial qB_{par_0} y recibimos un 1 → Pasamos al estado qB_{impar_0}
2. Estamos en el estado qB_{impar_0} y recibimos un 0 → Pasamos al estado qB_{impar_1}
3. Estamos en el estado qB_{impar_1} y recibimos un 1 → Pasamos al estado qB_{par_1}
4. Estamos en el estado qB_{par_1} y recibimos un 1 → Volvemos al estado qB_{impar_1}
5. Estamos en el estado qB_{impar_1} y recibimos un 1 → Volvemos al estado qB_{par_1}
6. Estamos en el estado qB_{par_1} y recibimos un 1 → Volvemos al estado qB_{impar_1}

- Terminamos de procesar la cadena y quedamos en el estado qB_{impar_1} (el cual no es estado de aceptación) → La cadena 101111 es rechazada por el autómata M_B.

Bibliografía

- Martello, S., & Toth, P. (1990). *Knapsack problems: Algorithms and computer implementations* (Sección 2.4, The Greedy Algorithm). Wiley.
- Dynkin, E. B. (1963). *The optimum choice of the stopping moment for a Markov process*. Soviet Mathematics Doklady, 4, 627–629.

- Ferguson, T. S. (1989). *Who Solved the Secretary Problem?* Statistical Science, 4(3), 282–289.

- Wikipedia. (2025). *Secretary problem*. Recuperado de https://en.wikipedia.org/wiki/Secretary_problem