TP2 - Teoría de Algoritmos

Universidad de Buenos Aires, Facultad de Ingeniería
Teoría de Algoritmos 75.29/95.06/TB024
Curso Echevarria
2do Cuatrimestre 2025

Grupo 10
Aylen R. Reynoso 100983
Bruno L. Starnone 108018
Victor A. Oliva Montaño 112748
Gonzalo E. Ranzani 105933
Huaman Quispe Bequer Smith 111540

Índice

PROBLEMA 1 - Programación Lineal..... 4
Supuestos..... 4
Limitaciones..... 4
Diseño..... 5
Variables de decisión..... 6
Restricciones..... 6
Pseudocódigo..... 8
Estructura de datos utilizada:..... 9
Análisis de Complejidad..... 10
Seguimiento..... 11
Tiempos de Ejecución..... 12
Resultados..... 12
Alternativas..... 13
PROBLEMA 2 - Redes de Flujo..... 15
Análisis..... 15
Supuestos y condiciones..... 15
Análisis de alternativas..... 15
Diseño..... 16
Adaptación de entrada..... 16
Salida del algoritmo..... 16
Pseudocódigo y explicación..... 18
Cuando no queden más caminos simples, se termina el algoritmo.
Estructuras de datos
Para una implementación, se asumen:..... 18
Seguimiento..... 20
Complejidad..... 23
Costo por iteración..... 24
Cantidad de iteraciones..... 24
Complejidad total..... 24
Solución..... 25
Informe de resultados..... 28
Fragmentación..... 28
¿PL o RF?..... 29
PROBLEMA 3 - Algoritmo de Aproximación..... 30
Análisis..... 30
Supuestos, condiciones, limitaciones y premisas..... 30
Análisis del tamaño del espacio de soluciones factibles del problema..... 32
Diseño..... 33
Seguimiento..... 34
Complejidad..... 35
Comparación con el tamaño del espacio de soluciones factibles..... 35
Sets de datos..... 37
Tiempos de Ejecución..... 37

Informe de Resultados....38
PROBLEMA 4 - Algoritmos Randomizados....39
Supuestos....39
Limitaciones....39
Premisa....39
Diseño....39
Propiedades....40
Grado de certeza....40
Pseudocódigo....41
Estructura de datos utilizadas....41
Seguimiento....42
Análisis de Complejidad....44
Grado de Certeza....45
Casos de Uso....46
Referencias....47

PROBLEMA 1 - Programación Lineal

Supuestos

En el modelo se asumieron los siguientes supuestos:

- **Red como grafo**
  La red de telecomunicaciones se representa como un grafo no dirigido:
  - Conjunto de nodos N={1,…,10}, que representan routers/switches de la facultad.
  - Conjunto de aristas E, donde cada {i,j}∈E es un enlace físico entre dos nodos.
- **Capacidades conocidas y constantes**
  Cada enlace {i,j} tiene una capacidad máxima C_{ij} (en MB) fija durante el período de análisis.
  No se consideran variaciones en el tiempo, congestión dinámica ni fallas de enlaces.
- **Fragmentación perfecta (TCP/IP)**
  El archivo de 10 MB puede fragmentarse en partes arbitrariamente pequeñas en cualquier nodo de la red.
  El protocolo TCP/IP garantiza la reconstrucción correcta de los fragmentos en el nodo destino (nodo 10).
- **Flujo estático**
  Se modela un flujo estacionario: no se consideran tiempos de transmisión ni latencias, sólo la cantidad total (MB) que circula por cada enlace.
- **Red sin pérdidas**
  No hay pérdidas de paquetes ni retransmisiones. Todo lo que sale del nodo 1 (origen) llega finalmente al nodo 10 (destino).
- **Enlaces bidireccionales**
  Cada enlace físico {i,j} es bidireccional. En el modelo se representa como dos arcos dirigidos (i,j) y (j,i) que comparten la misma capacidad total: $x_{ij} + x_{ji} \leq c_{ij}$.
- **Objetivo único: minimizar cantidad de enlaces activos**
  No se minimiza tiempo de transmisión ni uso de capacidad. El único criterio de optimalidad es usar la **menor cantidad posible de enlaces** para enviar los 10 MB.

Limitaciones

Las principales limitaciones del modelo planteado son:

- **Sin modelado de tiempo ni latencia**
  El modelo no distingue entre enlaces rápidos o lentos; dos rutas con misma cantidad de enlaces son equivalentes, aunque en la realidad tengan latencias muy diferentes.

- **Sin costos por ancho de banda**
  La capacidad se modela sólo como un límite máximo, pero no hay costo por "cercanía al límite" ni penalización por saturar enlaces.
- **Instancia de un solo flujo (unicommodity)**
  Sólo se envía un archivo desde el nodo 1 al nodo 10. No se considera tráfico simultáneo de otros usuarios o múltiples pares origen–destino.
- **Topología estática**
  No se consideran fallas de enlaces o variaciones de topología. Se supone que todos los enlaces están disponibles.
- **Escalabilidad teórica**
  El modelo usa variables binarias y es un problema de Programación Lineal Entera Mixta (PLEM).
  Para esta red pequeña es instantáneo, pero para redes muy grandes el tiempo de cómputo podría volverse significativo.

Diseño

Premisa del problema

Dada una red con 10 nodos y enlaces con capacidad limitada (en MB), se quiere determinar cómo fraccionar y enrutar un archivo de 10 MB desde el nodo 1 al nodo 10, usando la menor cantidad posible de enlaces de la red, aprovechando la capacidad de fragmentación de TCP/IP.

Conjuntos

- N: conjunto de nodos de la red.
  N={1,2,…,10}.
- E: conjunto de aristas no dirigidas.
  Cada arista $\{i, j\}$ tiene una capacidad ci_{ij} en MB.
- A: conjunto de arcos dirigidos, generado a partir de E:

$$A = (i, j), (j, i): i, j \in E.$$

Parámetros (constantes)

- c_{ij} : capacidad (MB) del enlace {i,j}∈E.
  Ejemplos:
  - $C_{1,2} = 5\,MB$
  - $C_{1,3} = 5\,MB$
  - $C_{2,6} = 4\,MB$
  - etc.
- bk : oferta/demanda de cada nodo k∈N, en MB:

- $b_1 = -10$: el nodo 1 envía 10 MB.
- $b_{10} = +10$: el nodo 10 recibe 10 MB.
- $b_k = 0$ para el resto de los nodos (nodos de tránsito).

Variables de decisión

- **Flujo por arco dirigido**

$$xij \geq 0 \forall (i,j) \in A$$

  - Mide la cantidad de MB que circula en el sentido i→j.
  - Tipo: continua, real no negativa.
  - Unidad: MB.

- **Uso del enlace**

$$y_{ij} \in \{0,1\} \forall \{i,j\} \in E$$

  - Y_{ij}=1: el enlace {i,j} se utiliza (hay flujo en al menos un sentido).
  - Y_{ij}=0: el enlace no se usa.
  - Tipo: binaria (indicadora).

Función objetivo

Minimizar la cantidad de enlaces utilizados:

$$\sum_{\{i,j\} \in E} yij$$

Restricciones

1. **Capacidad del enlace**

$$xij + xji \leq cij \cdot yij \; \forall \{i,j\} \in E$$

- Limita el flujo total por el enlace físico {i,j}.
- Si yij =0, fuerza $x_{ij} = x_{ij} = 0$

2. **Conservación de flujo en los nodos**

$$\sum_{i:(i,k) \in A} x_{ik} - \sum_{j:(k,j) \in A} x_{kj} = b_k \quad \forall k \in N$$

- Nodos intermedios (bk=0b_k=0bk=0): lo que entra es igual a lo que sale.
- Nodo 1: sale un flujo neto de 10 MB.
- Nodo 10: entra un flujo neto de 10 MB.

3. **Dominio de variables**

$$x_{ij} \geq 0 \quad \forall (i,j) \in A$$
$$y_{ij} \in \{0,1\} \forall \{i,j\} \in E$$

Pseudocódigo

```
# 1. Definir nodos y enlaces con capacidad
NODES     ← {1, 2, ..., 10}
EDGES_CAP ← diccionario {(i,j): capacidad_ij}
SUPPLY    ← diccionario {k: b_k}
            # b_1 = -10, b_10 = 10, resto 0

# 2. Construir arcos dirigidos a partir de las aristas
ARCS ← ∅
para cada (i,j) en EDGES_CAP:
   ARCS ← ARCS ∪ {(i,j), (j,i)}

# 3. Crear modelo MILP
prob ← nuevo ModeloPuLP(minimizar)

# 4. Variables de flujo x_ij (continuas >= 0)
para cada (i,j) en ARCS:
   definir x[i,j] continua ≥ 0

# 5. Variables binarias y_ij para uso de enlace
para cada (i,j) en EDGES_CAP:
   definir y[i,j] binaria ∈ {0,1}

# 6. Función objetivo
prob.objetivo ← sumatorio de y[i,j] para (i,j) en EDGES_CAP

# 7. Restricciones de capacidad
para cada (i,j) en EDGES_CAP:
   agregar restricción:
     x[i,j] + x[j,i] ≤ capacidad_ij * y[i,j]

# 8. Restricciones de conservación de flujo
para cada nodo k en NODES:
   inflow  ← suma de x[i,k] para todos (i,k) en ARCS
   outflow ← suma de x[k,j] para todos (k,j) en ARCS
   agregar restricción:
     inflow - outflow = SUPPLY[k]

# 9. Resolver el modelo con CBC
resolver(prob)
```

```
  # 10. Extraer solución
  para cada (i,j) en EDGES_CAP:
     si valor(y[i,j]) = 1:
        reportar flujo por ese enlace (x[i,j], x[j,i])

  # 11. Fin
fin algoritmo
```

Estructura de datos utilizada:

- Diccionarios para:
    - Capacidades de enlaces (EDGES_CAP).
    - Parámetros de oferta/demanda (SUPPLY).
    - Variables de decisión (x, y).
- Listas para:
    - Nodos (NODES).
    - Arcos dirigidos (ARCS).

Análisis de Complejidad

Complejidad del armado del modelo

Sea:

- $|N|$: cantidad de nodos.
- $|E|$: cantidad de aristas.
- $|A|=2|E|$: cantidad de arcos dirigidos.

En el código:

- Construcción de ARCS: recorre todas las aristas → $O(|E|)$.
- Creación de variables x (una por arco) → $O(|A|)=O(|E|)$
- Creación de variables y (una por arista) → $O(|E|)$.
- Restricciones de capacidad (una por arista) → $O(|E|)$.
- Restricciones de conservación (una por nodo, sumando sobre arcos incidentes) → $O(|A|)$

En total, el armado del modelo y sus restricciones es de orden:

$$O(|N|+|E|)$$

para la instancia dada.

Complejidad de resolución (solver)

- El problema es un MILP (Programación Lineal Entera Mixta).
- En términos teóricos, resolver un MILP es NP-difícil:
    - El tiempo de ejecución en el peor caso crece exponencialmente con el número de variables binarias (en este caso, $|E|$).
- En la práctica, para esta red:
    - 10 nodos,
    - 15 enlaces,
    - 15 variables binarias y ~30 variables de flujo,
- el solver CBC lo resuelve en tiempos muy pequeños (del orden de milisegundos).

Seguimiento

Para verificar que la solución obtenida por el modelo es coherente, se realiza el siguiente seguimiento conceptual:

1. **Balance global de flujo**

   - la suma de los b_k es:

   $$\sum_{k \in N} b_k = -10 + 10 = 0$$

   - Esto garantiza que el problema de flujo es potencialmente factible (lo que sale del sistema vuelve a entrar).

2. **Nodo origen (1)**

   - En la solución, el flujo que sale del nodo 1 suma 10 MB.
   - No entra flujo a este nodo.
   - Por lo tanto:
     $entra - sale = 0 - 10 = -10 = b1$
3. **Nodo destino (10)**

   - Llega un flujo total de 10 MB desde distintos vecinos.
   - No sale flujo desde el nodo 10.
   - Entonces:
     $entra - sale = 10 - 0 = 10 = b_{10}$

4. **Nodos intermedios**

   - Para cada nodo intermedio k, se verifica en la solución que:
     - La suma de los flujos que entran es igual a la suma de los flujos que salen.
     - Es decir, $entra - sale = 0 = b_k$.
   - Esto confirma que los nodos intermedios actúan solo como nodos de tránsito, sin crear ni consumir flujo.

5. **Enlaces usados/no usados**

   - Para los enlaces con $y_{ij} = 0$, los flujos x_{ij} y x_{ji} resultan numéricamente cero.
   - Para los enlaces con $y_{ij} = 1$, el flujo total cumple:

   $$x_{ij} + x_{ji} \leq c_{ij}$$

   - De esta manera, se respeta la capacidad de todos los enlaces.

Tiempos de Ejecución

- La prueba se ejecutó en un equipo de escritorio/notebook estándar.
- El tiempo total desde la construcción del modelo hasta la obtención de la solución es muy pequeño, del orden de fracciones de segundo (milisegundos).
- Este resultado es esperable porque:
    - La cantidad de nodos y enlaces es baja.
    - El número de variables binarias es reducido.
- En redes mucho más grandes, el tiempo podría crecer significativamente, pero en esta instancia la ejecución es prácticamente instantánea para el usuario.

Resultados

Objetivo

Evaluar la solución obtenida por el modelo de Programación Lineal Entera Mixta para el envío de un archivo de 10 MB desde el nodo 1 al nodo 10, verificando:

- El estado de la solución devuelta por el solver.
- El valor de la función objetivo (cantidad mínima de enlaces utilizados).
- La distribución del flujo sobre la red (fragmentación del archivo).
- El cumplimiento de las restricciones de capacidad y conservación de flujo en todos los nodos.

A partir de la resolución del modelo:

- Se obtiene una asignación de:
    - Variables y_{ij} que indican qué enlaces están encendidos (usados).
    - Variables x_{ij} que indican cuántos MB circulan por cada arco dirigido.

Interpretación general de la solución:

- Se logra enviar los 10 MB desde el nodo 1 al nodo 10 respetando todas las capacidades.
  El modelo decide un subconjunto mínimo de enlaces a utilizar, es decir:
    - No todos los enlaces disponibles de la red se activan.
    - Solo aquellos necesarios para armar uno o varios caminos que soporten, en conjunto, los 10 MB.

- Gracias a la posibilidad de fragmentar el archivo:

    - El flujo se puede repartir por distintos caminos.

    - Por ejemplo, una parte del archivo puede viajar por un camino que pasa por ciertos nodos intermedios y otra parte por un camino alternativo.

- El nodo 10 recibe la suma de todos los fragmentos, que totaliza 10 MB:
    - Esto garantiza la entrega completa del archivo.
    - A su vez, se cumple el objetivo de minimizar el número de enlaces utilizados.

Desde el punto de vista de la red:

- Algunos enlaces quedan apagados (no llevan tráfico) porque no son necesarios para lograr el objetivo con las capacidades disponibles.
- La solución respeta:
    - Conservación de flujo en todos los nodos.
    - Capacidad máxima de cada enlace.
    - Envío completo del archivo de 10 MB.

Metodología

Se formuló el problema como un modelo de Programación Lineal Entera Mixta y se implementó en Python utilizando la librería PuLP.

Se empleó el solver CBC, en modo minimización, con función objetivo igual a la suma de las variables binarias asociadas a los enlaces (cantidad de enlaces utilizados).

Una vez resuelto el modelo, se volcó la solución a un archivo de resultados (resultado_red.txt) que incluye:

- El estado reportado por el solver.
- El valor de la función objetivo.
- El detalle de cada enlace utilizado y el flujo que circula por él.
- El balance de flujo por nodo (flujo que entra, flujo que sale y diferencia entra–sale).

- A partir de esa información se construyeron:
    - **Tabla 1**: Enlaces utilizados y flujos (MB) en cada uno.

Enlace (i,j)	Flujo total (MB)	Flujo i→j (MB)	Flujo j→i (MB)
-1,2	5.00	5.00	0.00
-1,3	5.00	5.00	0.00
-2,6	3.00	3.00	0.00

-2,5	2.00	2.00	0.00
-3,4	3.00	3.00	0.00
-3,7	2.00	2.00	0.00
-4,8	3.00	3.00	0.00
-5,7	2.00	2.00	0.00
-6,9	3.00	3.00	0.00
-7,1	4.00	4.00	0.00
-8,1	3.00	3.00	0.00
-9,1	3.00	3.00	0.00

- **Tabla 2**: Balance de flujo por nodo.

#	Enlace	Flujo total (MB)
1	-1,2	5.00
2	-1,3	5.00
3	-2,6	3.00
4	-2,5	2.00
5	-3,4	3.00
6	-3,7	2.00
7	-4,8	3.00
8	-5,7	2.00
9	-6,9	3.00
10	-7,1	4.00
11	-8,1	3.00
12	-9,1	3.00

Resultados y observaciones

El solver CBC devolvió una solución óptima con estado *Optimal* y un valor de la función objetivo igual a 12.
Esto significa que el archivo de 10 MB puede enviarse desde el nodo 1 hasta el nodo 10 utilizando solamente 12 enlaces de la red, cumpliendo con el criterio de minimizar la cantidad de enlaces usados.

En la Tabla 1 se listan los enlaces efectivamente utilizados y el flujo que circula por cada uno. Se observa que:

- Desde el nodo origen (nodo 1) salen en total 10 MB, descompuestos en:
    - 5 MB por el enlace (1,2).
    - 5 MB por el enlace (1,3).
- El resto de los enlaces seleccionados conducen esos fragmentos a través de nodos intermedios hasta el nodo 10, respetando en todos los casos las capacidades máximas asignadas a cada enlace.
- No todos los enlaces disponibles de la red se activan: el modelo enciende únicamente aquellos estrictamente necesarios para transportar los 10 MB.

En la Tabla 2 se presenta el balance de flujo por nodo. Las principales observaciones son:

- El nodo 1 tiene un balance neto de –10 MB (sale 10 MB y no entra nada), por lo que actúa como origen del flujo (envía el archivo).
- El nodo 10 tiene un balance neto de +10 MB (entran 10 MB y no sale nada), actuando como destino del flujo (recibe el archivo completo).
- Los nodos intermedios (2 al 9) tienen balance neto cero (lo que entra es igual a lo que sale), por lo que funcionan exclusivamente como nodos de tránsito, cumpliendo la restricción de conservación de flujo.

La solución también puede interpretarse en términos de fragmentación del archivo en varios "sub–flujos" que recorren caminos distintos. Por ejemplo:

- 3 MB por un camino que pasa por nodos 3 y 4 hasta llegar a 10.
- 3 MB por un camino que pasa por nodos 2 y 6 hasta llegar a 10.
- 2 MB por un camino que pasa por nodos 2 y 5 hasta llegar a 10.
- 2 MB por un camino que pasa por nodos 3 y 7 hasta llegar a 10.

La suma de estos sub–flujos es exactamente 10 MB, y el protocolo TCP/IP se encarga de reconstruir el archivo en el nodo destino a partir de estos fragmentos.

Conclusión

La solución obtenida confirma que el modelo permite fragmentar y distribuir el archivo de 10 MB de manera eficiente en la red, utilizando sólo 12 enlaces y cumpliendo todas las restricciones de capacidad y de conservación de flujo. La interpretación de los resultados (enlaces activados, flujos por enlace y balances por nodo) muestra que:

- El archivo se reparte en varios caminos alternativos,
- El nodo 1 actúa como único emisor neto de los 10 MB,
- El nodo 10 recibe exactamente esos 10 MB como receptor neto,
  Los nodos intermedios sólo cumplen el rol de rutas de tránsito.

De esta manera, los resultados numéricos son coherentes con el comportamiento esperado del modelo y con el objetivo planteado de minimizar el número de enlaces utilizados para enviar el archivo desde el origen al destino.

Alternativas

Se pueden plantear variantes del modelo original según otros objetivos o restricciones:

1. **Cambiar la función objetivo**
    - Minimizar el costo total de los enlaces si cada enlace {i,j}tuviera un costo económico.
    - Minimizar el uso total de capacidad (suma de flujos) para reducir congestión.
    - Minimizar una combinación de:

        - Cantidad de enlaces.
        - Longitud de los caminos.
        - Costos.

2. **Modelo sin variables binarias**

    - Si el objetivo fuera simplemente encontrar cualquier enrutamiento factible (sin importar cuántos enlaces se usan), se podría formular solo con variables de flujo x_{ij}y sin y_{ij}.
    - Esto daría un problema de Programación Lineal más simple y rápido de resolver.

PROBLEMA 2 - Redes de Flujo

Análisis

Supuestos y condiciones

1. **Red como grafo con capacidades:** la red se modela como un grafo con nodos $N=\{1,\ldots,10\}$ y enlaces con capacidad $Cij \geq 0$ (en MB).
2. **Conversión a grafo dirigido:** como el grafo provisto es **no dirigido**, se lo transforma en un grafo dirigido reemplazando cada enlace *{i,j}* por dos arcos *(i,j)* y *(j,i)* con capacidad *Cij*. (En esta instancia el flujo neto va de 1→10, por lo que no se explota simultáneamente el enlace en ambos sentidos).
3. Capacidades enteras (MB): se asume que las capacidades son enteras, por lo que Ford–Fulkerson termina en una cantidad finita de aumentos.
4. **Fragmentación perfecta:** el archivo puede fragmentarse arbitrariamente, y el destino reconstruye el total recibido.
5. **Objetivo real del algoritmo usado:** Ford–Fulkerson garantiza flujo máximo. Para "minimizar enlaces", se aplica una regla heurística para elegir caminos de aumento (ver Diseño).
6. Ford–Fulkerson no está diseñado para minimizar cantidad de enlaces utilizados; solo maximiza flujo. Por lo tanto, la solución obtenida puede ser factible y transportar 10MB, pero no se puede asegurar optimalidad global en "#enlaces".
7. No se modelan tiempos/latencias, congestión dinámica ni fallas.
8. La resolución del ejercicio se hizo de forma manual y con diagramas, tal como lo permite el inciso 5, opción 1. Es por ello que el inciso 2c resulta no aplicable para el caso. Con el mismo criterio, la parte de Pseudocódigo será redactada en prosa con la explicaciones pertinentes de la resolución manual.

Análisis de alternativas

Se agrega esta sección debido a la incertidumbre sobre el algoritmo a utilizar para elegir los caminos. Dado que la resolución fue manual, se experimentaron con varias maneras de hallar los caminos de la fuente al sumidero. Los resultados destacables fueron que utilizar BFS da una solución peor que usando un algoritmo de maximización del mínimo cuello de botella.

Esto resulta contraintuitivo dado que usando BFS se busca minimizar la cantidad de aristas utilizadas, como pide la consigna, pero esto llevaba a usar un enlace extra. Por esta misma razón, se descartó el uso de BFS pero se quiere dejar constancia de que se experimentó con él como alternativa y fue descartado.

Diseño

Adaptación de entrada

El grafo original no está dirigido así que debe acomodarse para poder servir de entrada al algoritmo de Ford-Fulkerson, para cumplir con la condición 1 planteada en la sección anterior.

Entrada: lista de enlaces no dirigidos $\{i,j\}$ con capacidad c_{ij}.

Transformación: construir una red dirigida $G=(V,A)$ con:

- V=N
- $A=\{(i,j),(j,i) : \{i,j\} \in E\}$
- Capacidad residual inicial: $r_{ij} = c_{ij}$

Fuente: $s=1$

Sumidero: $t=10$

Para ello se hicieron los siguientes cambios, tomando como criterio que el nodo 1 es la **fuente**, y el nodo 10 el **sumidero**.

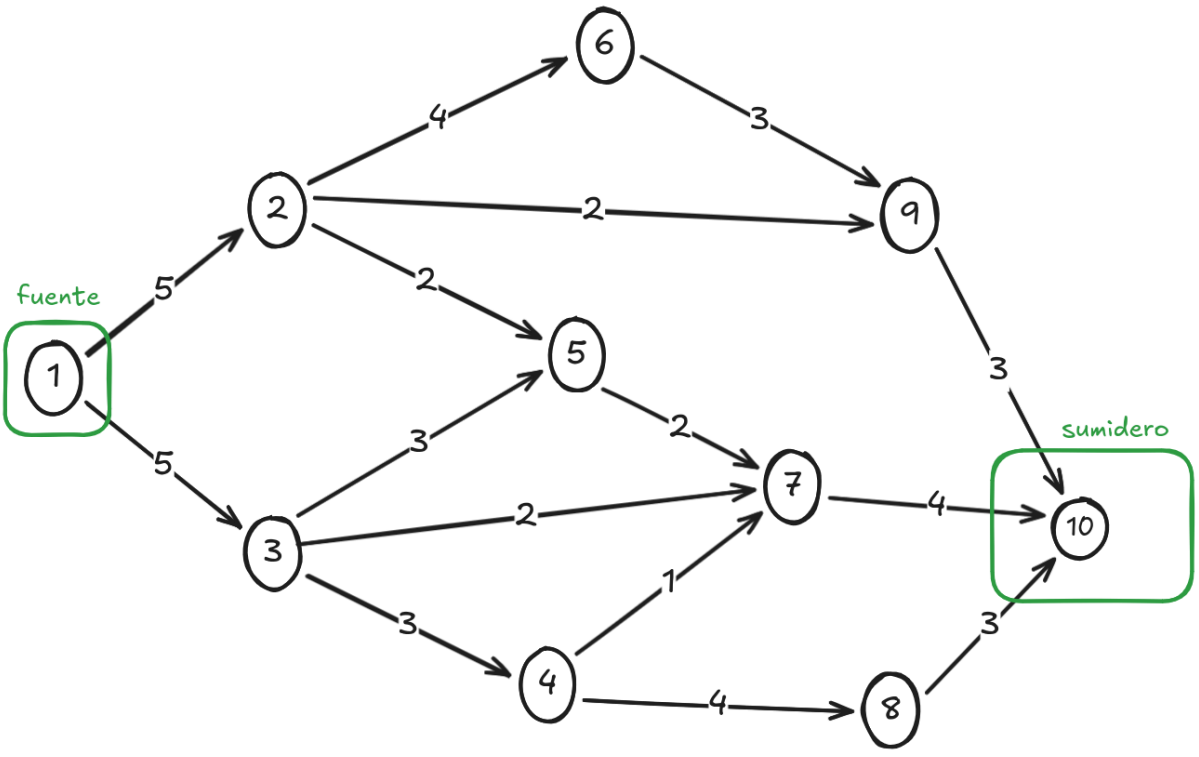


Figura 2.1: Grafo acondicionado para Ford-Fulkerson

Salida del algoritmo

- **Valor de flujo máximo *F*:** si $F \geq 10$, entonces existe un ruteo que permite enviar el archivo completo (factibilidad).

Flujos por arco / red residual: a partir del residual final, se obtiene cuánto "circula" por cada enlace. Esos valores se interpretan como **tamaños de fragmentos** que atraviesan los enlaces usados

Pseudocódigo y explicación

"Entrada: grafo no dirigido E con capacidades c_ij, fuente s=1, sumidero t=10
Salida: flujo f y conjunto de enlaces usados, más tabla de fragmentación

1) Convertir E → arcos dirigidos A (i,j) y (j,i) con capacidad c_ij
2) Inicializar flujo f_ij = 0 para todo arco; residual r_ij = c_ij

3) Mientras exista un camino aumentante P de s a t en el residual:
- Elegir P con criterio "maximizar el mínimo cuello de botella"
(widest path sobre r_ij)
- Δ = min{ r_ij : (i,j) ∈ P } (cuello de botella del camino)
- Para cada (i,j) ∈ P:
f_ij += Δ
r_ij -= Δ
r_ji += Δ (arco inverso en residual)
4) Devolver f y construir informe:
- Enlace usado si f_ij > 0 o f_ji > 0
- Fragmentación: descomponer el flujo en caminos s→t y asignar tamaños (MB)"

El algoritmo elige un **camino simple** para empezar a construir el grafo residual. El camino simple se encuentra bajo el criterio de maximización del mínimo cuello de botella. Entonces se recorre el grafo original buscando la arista saliente de la fuente con mayor capacidad. Luego seguir el camino de la misma, eligiendo siempre la mayor capacidad posible hasta llegar al sumidero. Una vez se llega al sumidero, se busca la mínima capacidad de las aristas y se actualiza el grafo residual en consecuencia.

Una vez actualizado el grafo residual, se comienza nuevamente con el mismo criterio. En caso de que un nodo tiene dos aristas salientes de igual capacidad, se desempata yendo al siguiente nodo con número menor. Es decir, si el nodo 2 tiene aristas de igual peso salientes para el nodo 3 y 4, se toma el 3 para continuar el camino.
Cuando no queden más caminos simples, se termina el algoritmo.

Estructuras de datos

Para una implementación, se asumen:

- **Lista de adyacencia** para representar el grafo dirigido.
- **Diccionarios / matrices** para capacidades c_{ij} , residual r_{ij} y flujo f_{ij} .

Seguimiento

Se muestra el algoritmo manual realizado con un set reducido.

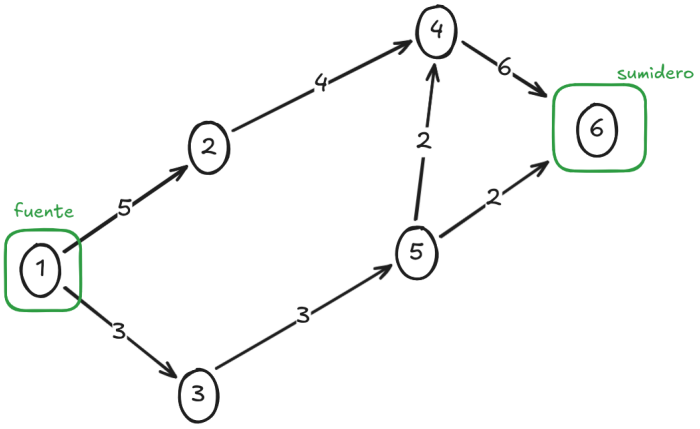


Figura 2.2: Ejemplo grafo reducido

1. Siguiendo el criterio de elección de camino, el camino elegido es el resaltado en azul.

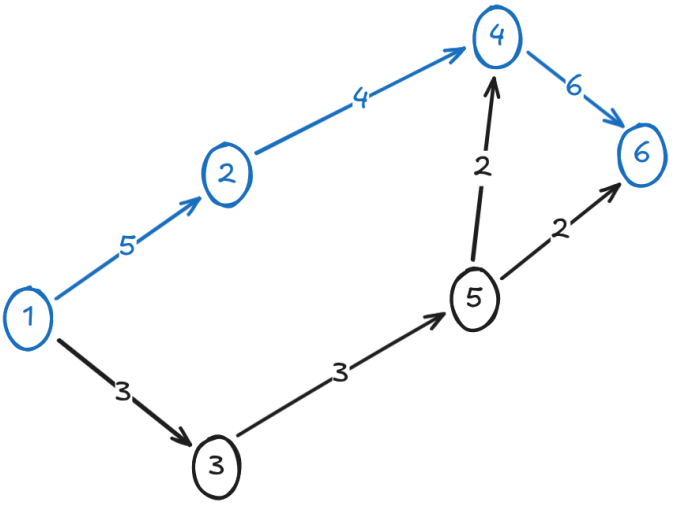


Figura 2.3: Paso 1 del algoritmo manual

2. Se actualizan los pesos con la mínima capacidad (4).

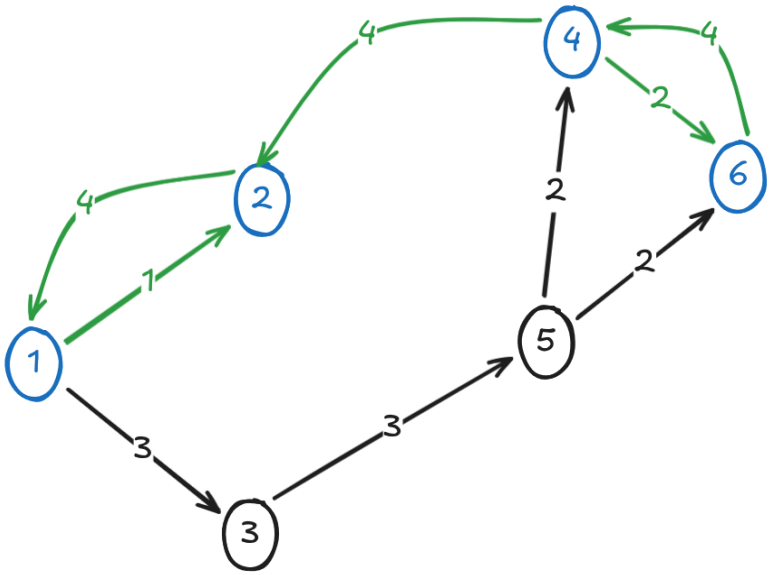


Figura 2.4: Actualización de grafo residual

3. El próximo camino es la arista que quedó con peso 3.

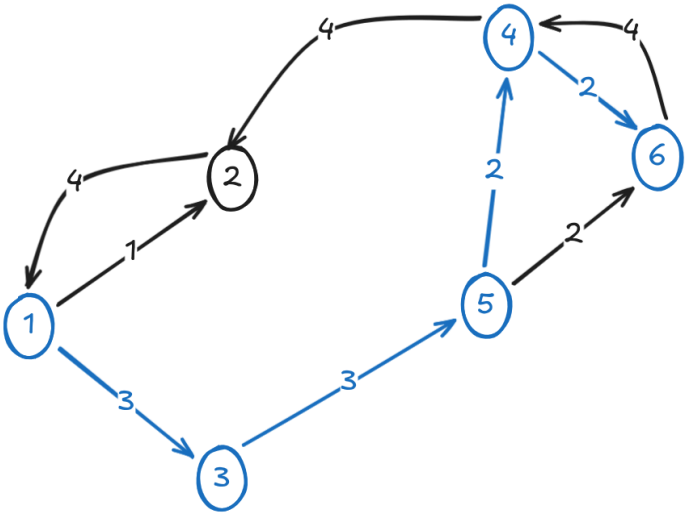


Figura 2.5: Elección del segundo camino

4. Se actualiza el grafo.

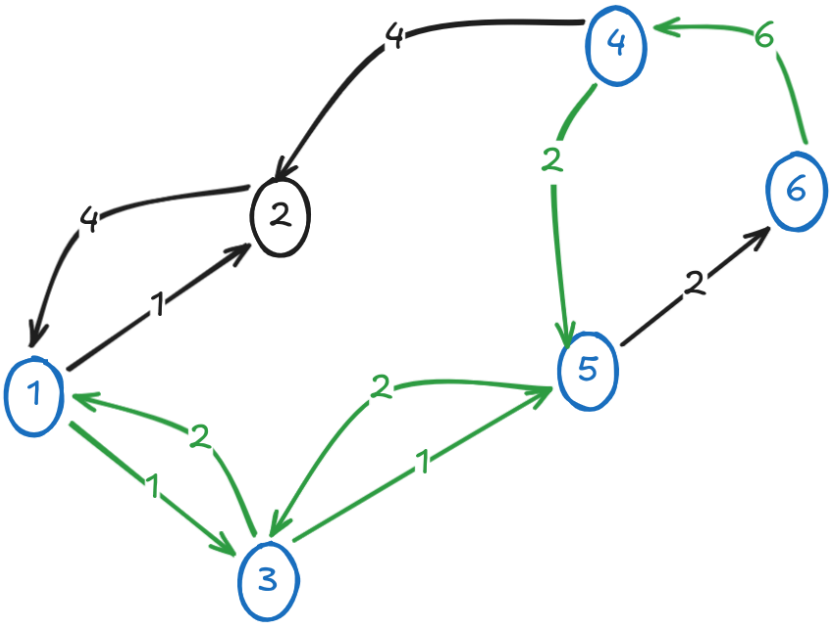


Figura 2.6: Actualización del grafo residual

5. Se elige el último camino simple que queda:

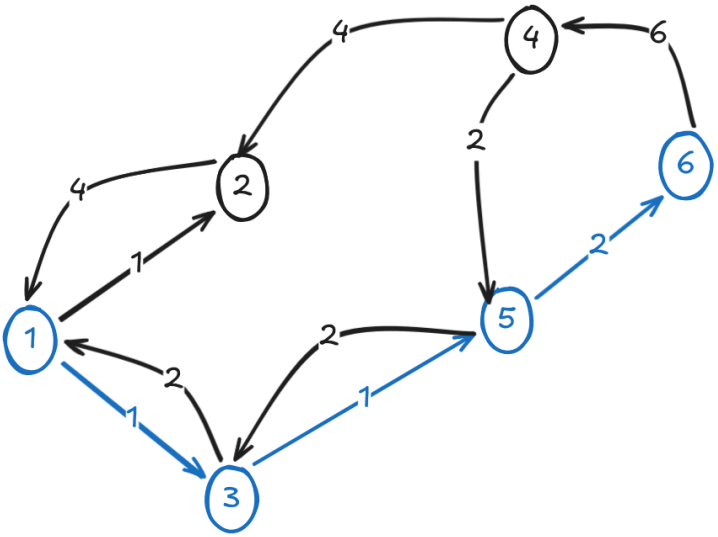


Figura 2.7: Elección del camino restante

6. Se actualiza el uso de la arista restante:

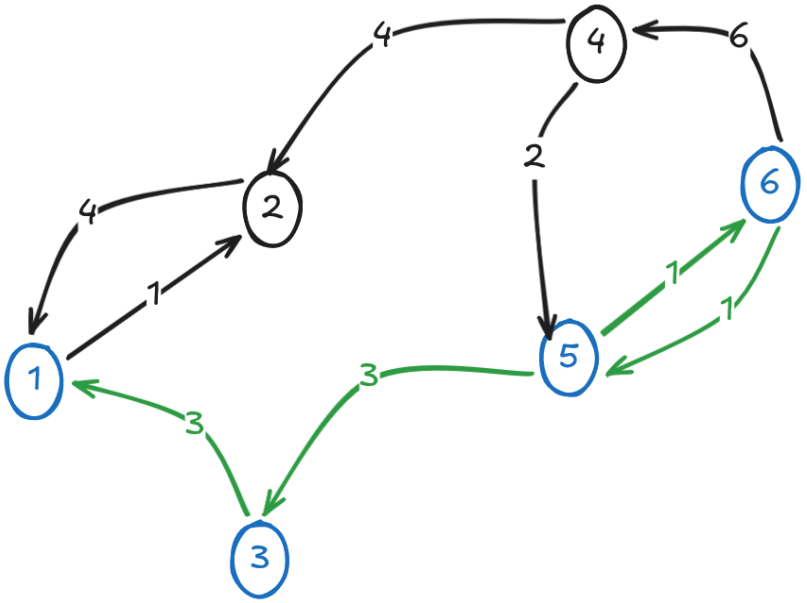


Figura 2.8: Actualización final de grafo residual

Finalmente se observa que el grafo residual dió un flujo máximo de 7 (suma de flujos en sumidero y fuente).

Complejidad

Sea G=(V,A) la red dirigida con $|V|$=n nodos y $|A|$=m arcos. Dado que el grafo original es no dirigido y se transforma en dirigido duplicando cada enlace *{i,j}* en *(i,j)* y *(j,i)*, se tiene típicamente $m \approx 2|E|$.

El procedimiento aplicado es el esquema de Ford–Fulkerson: en cada iteración se encuentra un camino aumentante en el grafo residual, se calcula el cuello de botella Δ (mínima capacidad residual del camino) y se actualiza la red residual restando Δ en los arcos usados y sumando Δ en los arcos inversos.

Costo por iteración

- **Búsqueda de un camino aumentante:** recorrer el grafo residual para construir un camino desde s a t cuesta $O(m)$ en el peor caso (exploración estándar de arcos y nodos, evitando ciclos).
- **Cálculo de Δ:** se recorre el camino hallado, costo $O(|P|) \leq O(n)$.
- **Actualización del residual:** también recorre los arcos del camino, costo $O(|P|) \leq O(n)$.

Entonces, el costo por iteración es:

$O(m+n)=O(m)$

Cantidad de iteraciones

Si las capacidades son **enteras**, cada aumento incrementa el flujo total al menos en 1 unidad. Por lo tanto, el número de iteraciones K queda acotado por el valor del flujo máximo F (muchos textos lo llaman C):

$K \leq F$ (o $K \leq C$)

Complejidad total

$T(n,m)=O(K \cdot m)=O(m \cdot F) \equiv O(mC)$

Nota práctica para este TP: como el objetivo es enviar **10 MB**, el flujo requerido está acotado ($F \leq 10$), por lo que en la práctica K es pequeño y el costo real queda muy bajo.

Solución

manual siguiendo el algoritmo descrito en las secciones anteriores.

1. Se elige el camino maximizando el mínimo cuello de botella.

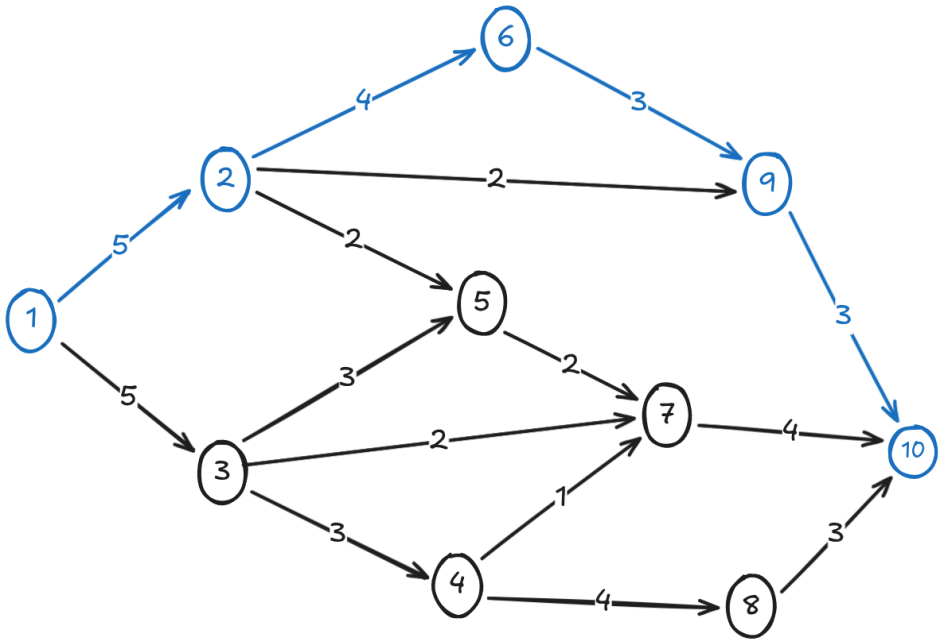


Figura 2.9: Elección del primer camino

2. Se actualizan los pesos.

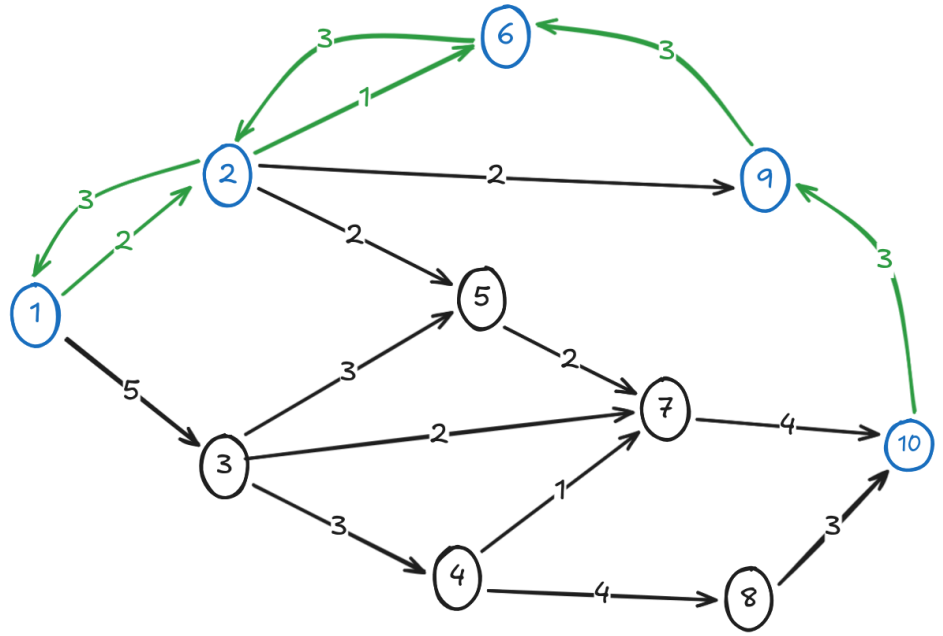


Figura 2.10: Actualización del grafo residual

3. Se busca el siguiente camino partiendo de la arista 1,3.

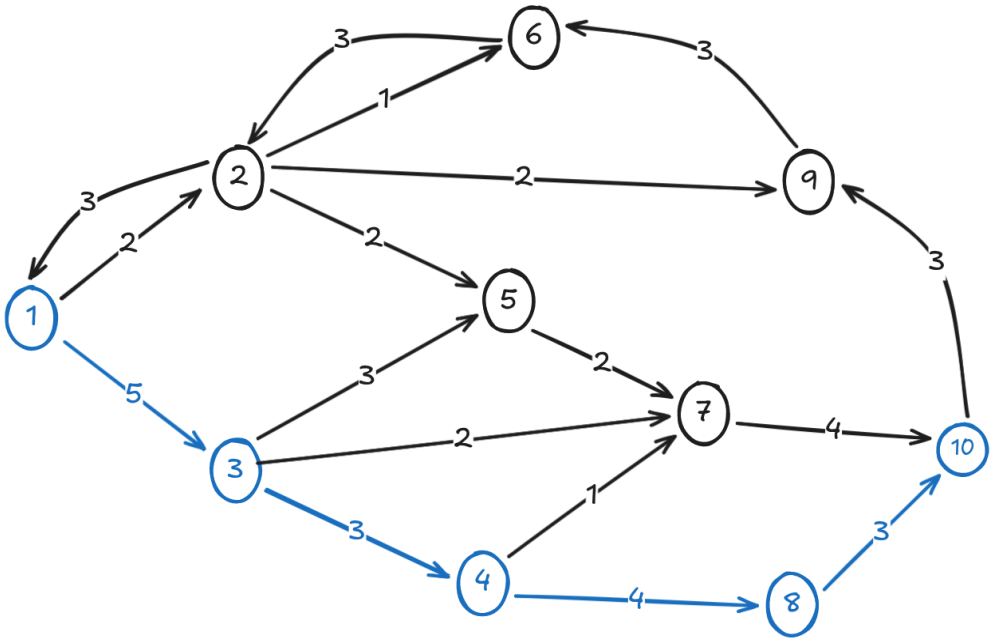


Figura 2.11: Elección del segundo camino

4. Se actualizan los pesos con la mínima capacidad (3)

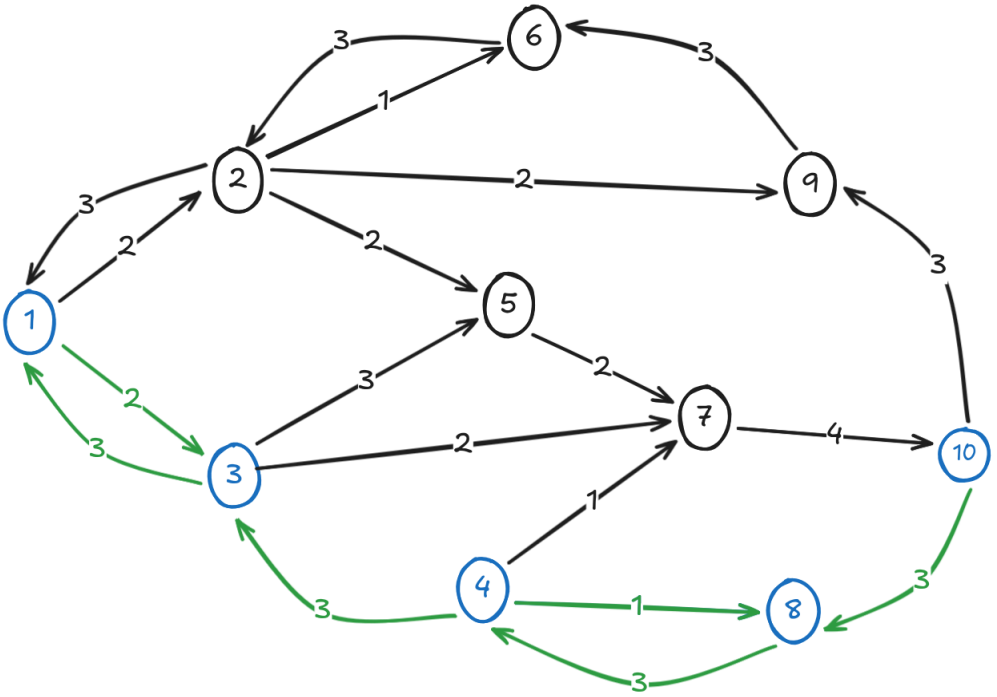


Figura 2.12: Actualización del grafo residual

5. El algoritmo sigue iterando bajo los mismos criterios (no se muestran ya que el seguimiento fue hecho en una sección anterior y se prioriza explicar la solución) hasta llegar a este grafo residual:

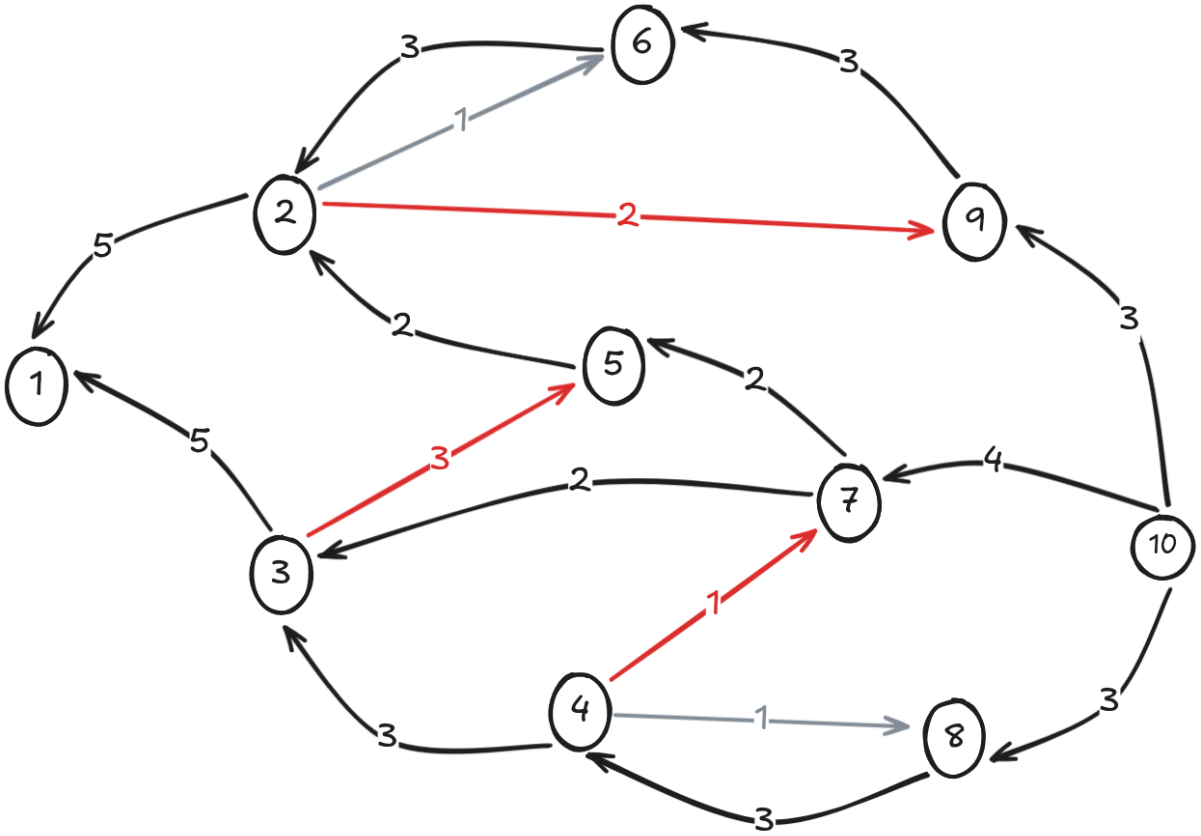


Figura 2.13: Grafo residual final.

Finalmente llegamos al resultado. La consigna solicita que se indique la mejor forma para fragmentar el archivo. Como solo se tiene el grafo residual, se pueden utilizar las aristas inversas para indicar cuántos MB deben viajar entre cada nodo. Por ejemplo, de la figura podemos ver que entre 1,3 irán 5MB que luego se separan en 2 MB entre 3 y 7; y en 3MB entre 3 y 4.

Asimismo se puede constatar que los 10MB solicitados salen de la fuente (5+5) y llegan al sumidero (3+4+3).

En el informe de resultados se redacta la fragmentación completa.

Informe de resultados

El grafo residual indica que el archivo se fragmenta en hasta 4 partes, utilizando 12 enlaces para realizar la comunicación.

Fragmentación

Enlace	**Fragmentos transportados - tamaño**
1,2	1 - 5 MB
1,3	2 - 5 MB
2,6	1.1 - 3 MB
2,5	1.2 - 2 MB
3,7	2.1 - 2 MB
3,4	2.2 - 3 MB
6,9	1.1 - 3 MB
9,10	1.1 - 3 MB
5,7	1.2 - 2 MB
4,8	2.1 - 3 MB
8,10	2.1 - 3 MB
7,10	1.2 y 2.1 - 4 MB

Tabla 2.1: Fragmentación obtenida del grafo residual

¿PL o RF?

- **Redes de Flujo**: excelente para verificar factibilidad y obtener un ruteo que logre F≥10F\ge 10F≥10.

- Pero minimizar la "cantidad de enlaces activos" es un objetivo combinatorio (costo fijo por usar enlace), que en general se modela mejor con Programación Lineal Entera/Mixta (PLEM/MILP).

- Por eso, aunque el residual de FF permite construir una fragmentación válida, no garantiza que use el mínimo número de enlaces; en cambio, PL/MILP puede incorporar directamente esa minimización.

PROBLEMA 3 - Algoritmo de Aproximación

NOTA:

Aclarando que para este problema usare como solución al algoritmo **NFD(Next Fit Decreasing),** y usare como una comparación al algoritmo **FFD(First Fit Decreasing)**.
Ambos algoritmos son de tipo greedy.
En algunos puntos de la consigna no usare el algoritmo de comparación por lo que solo lo usare donde pueda verse claramente la diferencia, como en el Análisis y Complejidad.

Bin Packing $\leq p$ ***NFD (Next Fit Decreasing)***

NOTA:
En la solución Bin Packing ***no llega a existir*** en el caso de que la lista esté vacía.

Análisis

<u>Supuestos generales (aplican a NFD y FFD)</u>

1. Capacidad fija del contenedor: **NDF** y **FFD** , ambos tienen contenedores de capacidad 1 (unidad).
2. Tamaños válidos: cada objeto tiene tamaño:
   $$0 < si < 1$$
   (si algún objeto $\geqq 1$, el problema no tendría solución).
3. Comparación exacta: **NDF** y **FFD** , en ambos se supone que las operaciones aritméticas de suma y comparación pueden realizarse sin errores significativos
4. Cada objeto debe estar exactamente en un contenedor y no puede partirse (no hay fraccionamiento como en el ejemplo de la mochila).
5. **NDF** y **FFD**, ambos algoritmos se puede decir que no tiene una optimalidad exacta, sino una que cumpla:
   $$NDF \leq 2.OPT$$
   $FFD \leqslant 1.22.OPT$ (la desigualdad es una aproximación)

Condiciones de funcionamiento para NFD (Next Fit Decreasing)

1. Antes de que empiece lo que hace el algoritmo requiere ordenar los objetos en orden **decreciente**.
2. Solo se permite colocar un objeto en el último contenedor abierto, esto quiere decir que no se puede revisar todos los contenedores como lo hace **FFD.**
3. Si el objeto no entra en ese contenedor, se abre un contenedor nuevo, a partir de este, se colocan los siguientes objetos mientras puedan caber dentro del contenedor.
4. **NFD** cumple con lo pedido de la consigna:

$$NDF \leq 2.OPT$$

Condiciones de funcionamiento para FFD (First Fit Decreasing)

1. Requiere ordenar los objetos en orden **decreciente**, igual que **NFD**.
2. Cada objeto se intenta ubicar en el **primer contenedor existente** donde entre.
3. Solo si no entra en ninguno, se abre un contenedor nuevo.
4. Este comportamiento produce soluciones mucho más compactas que **NFD**, pero con posible costo temporal mayor.
5. **FFD** cumple con lo pedido de la consigna y también se puede decir que es un poco mejor:

$$FFD \leqslant \frac{11}{9}.OPT + 1$$ -> aproximadamente **1.22**

Limitaciones (NFD y FFD)

1. Ninguno garantiza encontrar la solución óptima (el problema es NP-Hard).
2. Pueden producir soluciones subóptimas para ciertos casos específicos.
3. Con números reales, el comportamiento puede verse afectado por errores de punto flotante.
4. **FFD** puede tener un peor tiempo de ejecución **$O(n^2)$** si es en el peor caso, pero en un caso promedio es **O(nlog(n)).**
5. **NFD** puede desperdiciar capacidad en el contenedor al depender del principio *Next Fit*.

Premisas necesarias para la evaluación experimental

1. Los algoritmos deben recibir cualquier secuencia de objetos como entrada.
2. El orden inicial ***no*** afecta **NFD** ni **FFD** por el ordenamiento previo.

Análisis del espacio de soluciones

Para n objetos, cada objeto puede estar en cualquier contenedor existente o uno nuevo. En general, la cantidad de formas distintas de particionar un conjunto en subconjuntos no vacíos corresponde al número de particiones de Bell, Bn .

Las particiones de Bell crecen así:

$$B1 = 1,\ B2 = 2,\ B3 = 5,\ B4 = 15,\ B5 = 52, \ldots$$

Asintóticamente:

$$Bn \approx \left(\frac{n^{n}}{e^{n}.\sqrt{n}}\right)$$

Es exponencial, que es mucho mayor que algún algoritmo polinomial.

Conclusión

1. El espacio de soluciones posibles para el **Bin Packing** es exponencial.
2. **NFD** y **FFD** exploran solo una trayectoria determinista, lo cual es viable; explorar todo el espacio sería imposible para n grande.
3. La existencia de un espacio exponencial justifica el uso de algoritmos de aproximación.

Diseño

a. Pseudocódigo

```
def NFD(objetos):
    si la cantidad de objetos es menor a uno:
        devolver 0
    ordenamos de mayor a menor los objetos
    inicializo un empaquetamiento con un contenedor vacío y un limite 1

    para cada objeto de los objetos ordenados hacer:
        tamaño y contenedor es igual al último contenedor creado

        si el objeto es menor o igual al tamaño:
            decrementamos el límite con el objeto
            agregamos al contenedor el objeto
        si no:
            inicializamos un nuevo contenedor agregando el objeto y
            actualizamos el límite
```

devolvemos la cantidad de contenedores usados

b. Mostrar cómo se cumple con la garantía solicitada (se pueden usar citas y referencias)

Wikipedia. (s.f.). *Bin packing problem*. En Wikipedia. Obtenido en noviembre de 2025, de *https://en.wikipedia.org/wiki/Bin_packing_problem*

Para ser mas exacto esta en la sección:

- Heurística en línea:
  - Algoritmos de clase única **(Next Fit).**
    "es lo mismo a decir **(Next Fit Decreasing)"**

Wikipedia. (s.f.). *Next-fit bin packing*. En Wikipedia. Obtenido en noviembre de 2025, de https://en.wikipedia.org/wiki/Next-fit_bin_packing

También se encuentra en esta referencia.

c. Detallar las estructuras de datos utilizadas. Justificar su elección.

- Para representar el empaquetamiento utilizamos una **lista de pares**:
  [capacidad_restante, lista_de_objetos]
  ya que esto nos permite:
  1. Acceder y modificar en tiempo constante **O(1)**
  2. Insertar nuevos contenedores en **O(1)**
  3. Y al recorrer secuencialmente con costo lineal
- Utilizamos la función **sorted(objetos, reverse=True)** que implementa **Timsort,** además tiene una complejidad **O(n.log(n))**.

  ya que esto nos permite:
  1. Ordenar la lista de objetos de mayor a menor.
  2. Esta función nos ayuda a que tenga un recorrido secuencial y nunca volvamos atrás.

Seguimiento

Ejemplo para el seguimiento:

- objetos = [0.37, 0.19, 0.48, 0.42, 0.25, 0.34, 0.27, 0.36]

1. verificamos que la cantidad no sea menor a 1 -> falso
2. ordenamos los objetos de mayor a menor:
    a. objetos_ordenados = [0.48, 0.42, 0.37, 0.36, 0.34, 0.27, 0.25, 0.19]
3. inicializamos un contenedor vacío con un limite 1
4. recorremos cada objeto ordenado

    objeto = 0.48
    a. tamaño = 1.0 y contenedor = vacío
    b. como el 0.48 <= 1.0
    c. actualizamos el tamaño -> tamaño = 0.52
    d. agregamos el objeto al contenedor -> contenedor = [0.48]

    objeto = 0.42
    a. tamaño = 0.52 y contenedor = [0.48]
    b. como el 0.42 <= 0.52
    c. actualizamos el tamaño -> tamaño = 0.1
    d. agregamos el objeto al contenedor -> contenedor = [0.48, 0.42]

    objeto = 0.37
    a. tamaño = 0.1 y contenedor = [0.48, 0.42]
    b. como el 0.37 > 0.1
    c. creamos un nuevo contenedor y actualizamos
    d. tamaño = 1.0 - 0.37 -> tamaño = 0.63 y contenedor = [0.37]

    objeto = 0.36
    a. tamaño = 0.63 y contenedor = [0.37]
    b. como el 0.36 <= 0.63
    c. actualizamos el tamaño -> tamaño = 0.27
    d. agregamos el objeto al contenedor -> contenedor = [0.37, 0.36]

    objeto = 0.34
    a. tamaño = 0.27 y contenedor = [0.37, 0.36]
    b. como el 0.34 > 0.27
    c. creamos un nuevo contenedor y actualizamos
    d. tamaño = 1.0 - 0.34 -> tamaño = 0.66 y contenedor = [0.34]

    objeto = 0.27
    a. tamaño = 0.66 y contenedor = [0.34]
    b. como el 0.27 <= 0.66
    c. actualizamos el tamaño -> tamaño = 0.39
    d. agregamos el objeto al contenedor-> contenedor = [0.34, 0.27]

    objeto = 0.25
    a. tamaño = 0.39 y contenedor = [0.34, 0.27]
    b. como el 0.25 <= 0.39
    c. actualizamos el tamaño -> tamaño = 0.14

d. agregamos el objeto al contenedor -> contenedor = [0.34, 0.27, 0.25]

objeto = 0.19

a. tamaño = 0.14 y contenedor = [0.34, 0.27, 0.25]
b. como el 0.19 > 0.14
c. creamos un nuevo contenedor y actualizamos
d. tamaño = 1.0 - 0.19 -> tamaño = 0.81 y contenedor = [0.19]

5. devolvemos la cantidad de contenedores usados = 4

Pero el óptimo es:

[0.48, 0.27, 0.25], [0.42, 0.36, 0.19], [0.37, 0.34] -> OPT = 3

Complejidad

Ordenamiento es: **O(n.log(n))**
Recorrido de los objetos ordenados: **O(n)**
Entonces: **T(n) = O(n.log(n))**

dato: si se puede ver en el archivo First_Fit_Decreasing.py es casi lo mismo solo que en ese caso hace otro recorrido de los contenedores que se creador y verifica en qué contenedor cabe el objeto.
Haciendo que que este algoritmo en el peor caso tenga complejidad **O(n²)**

Comparación con el tamaño del espacio de soluciones factibles

El problema de Bin Packing tiene un espacio de soluciones factibles extremadamente
grande.

Cada solución corresponde a una forma distinta de asignar los n objetos a contenedores sin exceder la capacidad.

El número total de formas de particionar un conjunto de n elementos está dado por los
números de Bell, que crecen de manera **super-exponencial**:

$$Bn \approx \left(\frac{n^{n}}{e^{n}.\sqrt{n}} \right)$$

Esto significa que **explorar exhaustivamente todas las soluciones posibles es completamente inviable incluso para valores pequeños de n**, lo que explica por qué el problema es **NP-Hard**.

Frente a ese espacio de soluciones gigantesco, el algoritmo aproximado utilizado

(Next-Fit-Decreasing) tiene una complejidad:

$$T(n) = O(n.log(n))$$

Es decir, **crece de forma muy lenta** comparado con el número total de soluciones
posibles.

El algoritmo solo calcula una **única solución aproximada**, sin explorar

combinaciones ni realizar búsquedas exhaustivas.

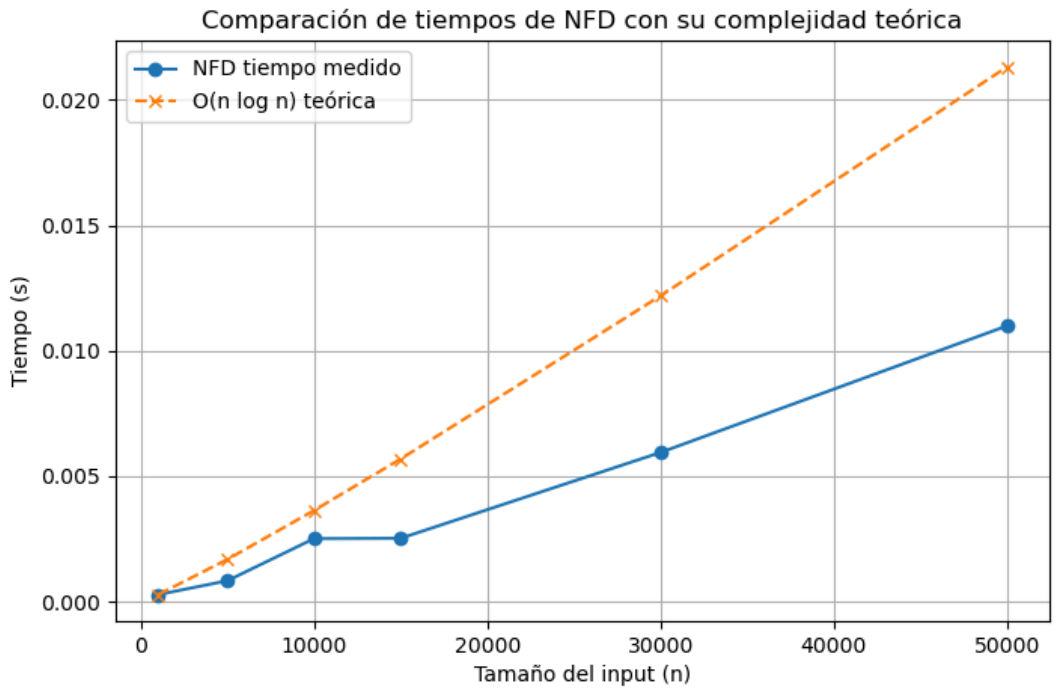
En consecuencia:

- El algoritmo **es eficiente** en relación con el tamaño del espacio de soluciones.
- Es razonable usarlo incluso para **tamaños grandes de entrada**, donde un enfoque exacto sería imposible.

Sets de datos

Para evaluar el rendimiento del algoritmo se utilizan arreglos de números flotantes de diversos tamaños generados de manera aleatoria y que cada objeto o dato del arreglo está en el intervalo (0,1). Adjunto en la carpeta ***sed_de_datos_NFD.py***

Tiempos de Ejecución



1000 elementos: 0.00027222s
5000 elementos: 0.00082793s
10000 elementos: 0.00250636s
15000 elementos: 0.00251968s
30000 elementos: 0.00594292s
50000 elementos: 0.01099172s

Informe de Resultados

a. Redactar un informe de resultados comparando los tiempos de ejecución con la complejidad temporal.

Objetivo:

Evaluar los tiempos de ejecución del algoritmo Next-Fit Decreasing (**NFD**) y compararlos con su complejidad teórica **O(n log n)**.

Metodología:

- Se generaron conjuntos de datos aleatorios de distintos tamaños: 1 000; 5 000; 10 000; 15 000; 30 000; 50 000.
- Cada tamaño se ejecutó varias veces y se promediaron los resultados para suavizar variaciones aleatorias.
- Se midió el tiempo de ejecución de **NFD** y se comparó con la curva teórica **O(n log n)**, escalada para facilitar la comparación visual.

Resultados y Observaciones:

- Los tiempos de ejecución crecen aproximadamente siguiendo la forma **O(n log n)**.
- Los tiempos medidos aparecen por debajo de la curva teórica porque esta representa el **peor caso**, y los sets de datos aleatorios no lo generan.
- A medida que aumenta el tamaño de input, los tiempos medidos siguen la **tendencia de crecimiento O(n log n)**.
- Se puede mencionar como referencia que otros algoritmos como FFD tienen **mayor complejidad O(n^2)**, aunque en este informe no se grafican ni se analizan en detalle.

Conclusión:

El algoritmo **NFD** cumple con su complejidad teórica **O(n.log(n))** en promedio. La comparación entre los tiempos medidos y la curva teórica permite observar claramente la tendencia de crecimiento, cumpliendo con los objetivos del informe.

b. El gráfico comparativo de tiempos debe incluir tanto la curva con los valores medidos como la curva correspondiente a la complejidad temporal determinada.

El gráfico comparativo se genera con el código entregado en los sets de datos, mostrando los tiempos medidos de **NFD** frente a la curva teórica **O(n log n)**. Esto permite visualizar cómo el algoritmo sigue la tendencia de crecimiento esperada.

PROBLEMA 4 - Algoritmos Randomizados

Supuestos

- El verificador (Victor) es honesto: sigue el protocolo, elige el desafío al azar en cada ronda y verifica la respuesta sin intentar hacer trampa.
- Las dos pelotas son idénticas en todo aspecto excepto en el color.
- Las rondas del protocolo son independientes entre sí. En cada ronda, el verificador decide de forma aleatoria e independiente si intercambia o no las pelotas.
- El canal de comunicación es confiable: no hay errores al transmitir los desafíos ni las respuestas.

Limitaciones

- El protocolo no alcanza nunca un grado de certeza del 100% en un número finito de rondas.
- Aumentar la cantidad de rondas incrementa la certeza, pero también el tiempo de interacción entre Peggy y Victor.

Premisa

Peggy (prover) tiene dos pelotas de colores distintos (una roja y una verde) y puede distinguir perfectamente sus colores. Victor es daltónico rojo/verde y no puede distinguirlas visualmente.

Peggy quiere convencer a Victor de que realmente puede distinguir las pelotas, sin revelar cuál es la roja y cuál es la verde, es decir, sin dar ninguna información extra sobre los colores más allá de la veracidad de la afirmación “puedo distinguirlas”.

Diseño

Se implementa un protocolo Interactive Zero Knowledge Proof el cual consta de N rondas entre Prover y Verifier donde:

En cada ronda se realizará un Commitment: ambas partes saben la posición inicial de las pelotas (mano izquierda, mano derecha).

El Verifier genera un Challenge random, cambiar o no las pelotas de posición, y se lo envía al Prover para que lo resuelva.

El Prover responde si las posiciones han sido intercambiadas o no (True o False)

Si la respuesta es incorrecta entonces el Verifier puede asegurar que el Prover no tiene la capacidad de distinguir colores.

Si por el contrario, la respuesta es correcta, el Verifier acepta la respuesta con un 50% de validez ya que existe un 50% de probabilidad de que el Prover haya adivinado la respuesta correcta.

Al pasar las n rondas la probabilidad de que el Prover acierte adivinando va disminuyendo.
Se determina N tal que el grado de certeza sea considerable.

El protocolo pertenece a la familia de algoritmos randomizados Monte Carlo

Propiedades

Este protocolo satisface las tres propiedades principales:

- Completitud:

  Si Peggy puede distinguir efectivamente los colores de las pelotas (prover honesto), entonces en cada ronda puede determinar con certeza si Victor intercambió o no las pelotas.

  Su respuesta será siempre correcta, por lo que, en ausencia de errores de comunicación, Victor aceptará el protocolo luego de las N rondas.

  Es decir, cuando la afirmación es verdadera y el prover es honesto, la probabilidad de que el verificador acepte es 1.
- Solidez:

  Si el challenge no fue resuelto correctamente el Verifier puede determinar que el prover no dice la verdad, es decir es poco probable que el Verifier sea engañado por el Prover

  Esta probabilidad decrece exponencialmente con NNN. Por lo tanto, un prover deshonesto solo puede engañar al verificador con una probabilidad cada vez más pequeña a medida que se incrementa la cantidad de rondas.
- Conocimiento Cero:

  El Verifier en ninguna instancia gana nuevo conocimiento (color de las pelotas), solo el hecho de si la resolución es válida o no.

La secuencia de pares (challenge, respuesta) que se obtiene cuando Peggy es honesta puede ser simulada por un verificador que no tiene acceso a las pelotas: basta con generar challenges al azar y suponer que "Peggy responde siempre bien".

Como el protocolo no revela ninguna información adicional sobre los colores de las pelotas, sino únicamente la veracidad de la afirmación "Peggy puede distinguirlas", se cumple la propiedad de conocimiento cero.

Grado de certeza

A continuación se determina N tal que el Verifier tenga al menos un 90% de certeza de que el statement es verdadero.

Probabilidad de que Peggy engañe a Victor en una ronda: P = ½ y 1 - P = ½
Cada ronda es independiente.

Probabilidad de que Peggy pueda engañar en N rondas es:

$$P_{engaño}(N) = \left(\frac{1}{2}\right)^N$$

Probabilidad de que Peggy NO pueda engañar en N rondas es:

$$C(N) = 1 - P_{engaño}(N) = 1 - \left(\frac{1}{2}\right)^N$$

Cuanto más rondas se realicen, mayor será la certeza que tendrá el verifier

El grado de certeza entonces está dado por Certeza = $1 - \left(\frac{1}{2}\right)^N$

Queremos: Certeza ≥ 0.90

$1 - \left(\frac{1}{2}\right)^N \geq 0.90$

$\left(\frac{1}{2}\right)^N \leq 0.10$

$log\left(\left(\frac{1}{2}\right)^N\right) \leq log\,(0.10)$

$N \,.\, log\left(\frac{1}{2}\right) \leq -1$

$N \geq \frac{1}{log\left(\frac{1}{2}\right)}$

$N \geq 3.32$

Entonces N entero debe ser como mínimo 4 para asegurar que Victor pueda detectar el engaño en un 90%

Pseudocódigo

```
Protocolo colorblind_ZK(rondas):

  Estado inicial = [color1 en pos1, color2 en pos2]

  Para cada i en el rango 1..Rondas:
    Estado ronda = Estado inicial # Establecer commitment

    acción verifier= Elegir_random(1,0) # 1 intercambiar, 0 no intercambiar

    Nuevo estado = intercambiar(estado ronda, acción verifier) #Crear Challenge
    respuesta Prover = determinar_intercambio(Nuevo estado)

    Si respuesta Prover != acción Verifier:

      Retornar “Inválido”

   Retornar “Válido”
```

Estructura de datos utilizadas

- Arreglo de 2 elementos para mantener las posiciones de las pelotas

Seguimiento

Para N = 3 prover honesto:

Primera ronda:

Commitment = [Verde , Rojo]

Challenge = Intercambiar

Respuesta = Las bolas fueron intercambiadas de posición

Verificación = Válido

Certeza = 50%

Segunda ronda:

Commitment = [Verde , Rojo]

Challenge = No intercambiar

Respuesta = Las bolas no fueron intercambiadas de posición

Verificación = Válido

Certeza = 75%

Tercera ronda:

Commitment = [Verde , Rojo]

Challenge = Intercambiar

Respuesta = Las bolas fueron intercambiadas de posición

Verificación = Válido

Certeza = 87.5%

Para N = 4 prover deshonesto:

Primera ronda:

Commitment = [Verde , Rojo]

Challenge = Intercambiar

Respuesta = Las bolas fueron intercambiadas de posición

Verificación = Válido

Certeza = 50%

Segunda ronda:

Commitment = [Verde , Rojo]

Challenge = No intercambiar

Respuesta = Las bolas no fueron intercambiadas de posición

Verificación = Válido

Certeza = 75%

Tercera ronda:

Commitment = [Verde , Rojo]

Challenge = No intercambiar

Respuesta = Las bolas no fueron intercambiadas de posición

Verificación = Válido

Certeza = 87.5%

Cuarta ronda:

Commitment = [Verde , Rojo]

Challenge = Intercambiar

Respuesta = Las bolas no fueron intercambiadas de posición

Verificación = Inválido

Certeza = 0%

Análisis de Complejidad

Función colorblind_ZK:

- Inicialización estado → O(1)
- Bucle rondas → O(N)
    - Elegir aleatoriamente → O(1)
    - Intercambiar elemento → O(1) (swap 2 elementos)
    - Comparar estado → O(1) (comparar 2 arrays de 2 elementos)

Complejidad total: O(N), lineal respecto a la cantidad de rondas.

Grado de Certeza

Se puede determinar el grado de certeza de la prueba a partir de la cantidad de rondas realizadas. A medida que se van ejecutando las rondas, la probabilidad de que Peggy resuelva correctamente el challenge de Victor solamente adivinando es cada vez más baja. Por consiguiente, la certeza es cada vez más alta. Este comportamiento se puede observar en la Figura 1.

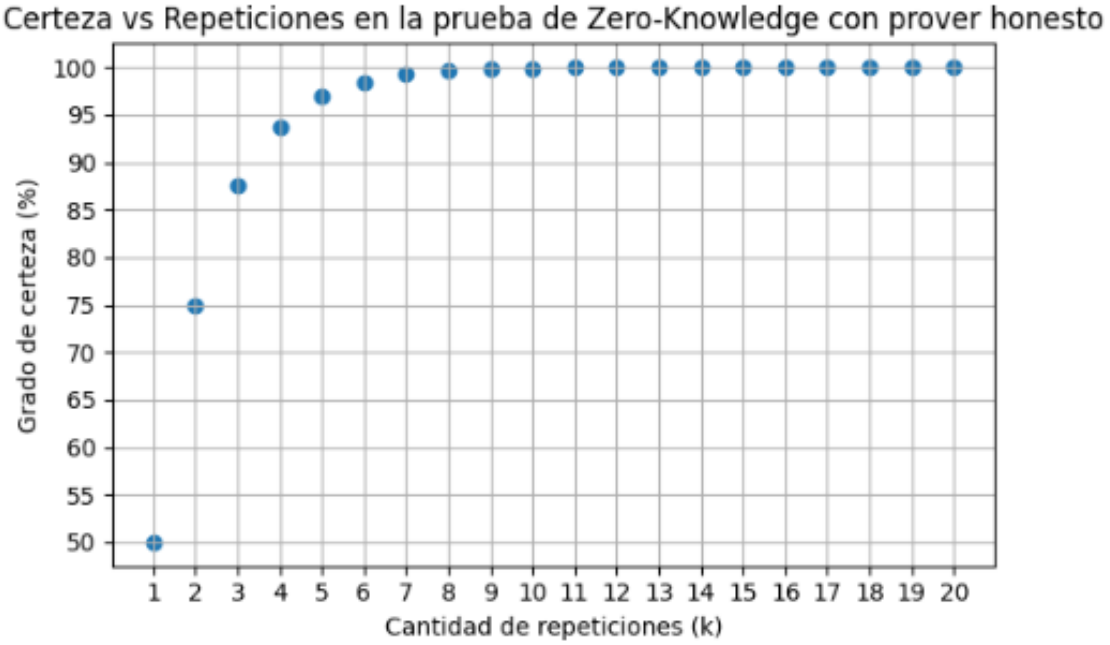


Figura 4.1: Grado de certeza vs cantidad de repeticiones

Como se mencionó previamente, el grado de certeza del verifier depende de la cantidad
de rondas realizadas. La fórmula que describe este comportamiento es la siguiente:

$$Certeza = 1 - \left(\frac{1}{2}\right)^{N}$$

La Tabla 1 indica cuál es el grado de certeza al realizarse diferentes cantidades de rondas:

Cantidad de rondas	Grado de certeza
1	50%
2	75%
3	87.5%
4	93.75%
5	96.875%
6	98.4375%
7	99.21875%

Tabla 4.1: Grado de certeza para diferentes cantidades de repeticiones

Se observa que al verifier le alcanza con 6 rondas para garantizar casi con total seguridad que Peggy no miente. Sin embargo, es importante recalcar que nunca se llegará al 100% de certeza, ya que solo sería posible si la cantidad de rondas tendiera a infinito, y eso no se puede llevar a cabo.

Casos de Uso

Infraestructura Blockchain:

Layers 2 como ZKSync utilizan ZK rollups para escalar Ethereum.

ZKSync agrupa múltiples transacciones en un solo batch, estas son procesadas off-chain y asociadas a ZK-SNARKS. Estas pruebas, que demuestran matemáticamente la validez de todas las transacciones del batch, son enviadas a la Layer 1 donde son validadas por un contrato verificador y asociadas a un transacción on chain heredando la seguridad de Ethereum reduciendo costos y tiempos de ejecución.

Autenticación:

Los métodos tradicionales de autenticación requieren que el usuario revele información sensible como datos personales, contraseñas, etc.

Soluciones como ZK Passport permiten a un usuario demostrar la posesión de un documento o el cumplimiento de un requisito (ej. ser mayor de edad) mediante una prueba criptográfica. El sistema de verificación válida la prueba sin jamás acceder o almacenar el dato sensible en sí, garantizando la privacidad absoluta del usuario.

Referencias

[1] Curso Echevarría (2025, Noviembre). *Redes de Flujo I.pptx - Página 14.* https://docs.google.com/presentation/d/1-HPGnihUB9BA9iWEx1qa9F7MbHK4tTsy/edit?slide=id.g351ea8895b6_0_57#slide=id.g351ea8895b6_0_57

Cem Dilmegani. (2025, Noviembre). *Zero-Knowledge Proofs: How it works & use cases*. AI Multiple. Obtenido el 28 de noviembre del 2025 de https://aimultiple.com/zero-knowledge-proofs