Resolución de Tarea 1 - Complejidad Algorítmica y Análisis Amortizado (Fecha: 29 de Septiembre de 2025)

Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI5651 - Diseño de Algoritmos I
Septiembre - Diciembre 2025
Estudiante: Junior Miguel Lara Torres (17-10303)

Tarea 1 (9 puntos)

Indice

Pregunta 1

Consideraciones

Teniendo en cuenta los siguientes puntos

- $T_{permutaciones} \in \Theta(1)$
- $T_{ordenado} \in \Theta(n)$
- El arreglo **a** puede tener la propiedad de:
  - Un **multiconjunto** teniendo así una cantidad de permutaciones igual a
$$\binom{n}{m_1, m_2, ..., m_k} = \frac{n!}{m_1!m_2!...m_k!}$$
  - Un **conjunto** teniendo asi una cantidad de permutaciones igual a $n!$

- Para el punto 3, se tiene que, en cualquier caso, se cumple que:

$$\frac{n!}{m_1!m_2!...m_k!} \leq n!$$

Esto porque, cuando no hay repetición de elementos (conjunto), entonces se cumple $(\forall k | 1 \leq k \leq n : m_k = 1)$, por lo tanto:

$$\frac{n!}{1!1!...1!} \leq n! \iff n! \leq n!$$

En caso de haber al menos una repetición de un elemento, entonces:

$$\frac{n!}{m_1!m_2!...m_k!} < n!$$

Peor caso

Con el arreglo **a** teniendo propiedades de conjunto, tendremos $n!$ permutaciones para revisar. Adicionalmente, el predicado **ordenado** tarda $O(n)$, se tiene entonces $f = n! \cdot n)$, por lo tanto, $bogoMin \in O(n! \cdot n)$.

Mejor caso

Teniendo un arreglo **a** con propiedades de conjunto y la primera permutación sea la ordenada, y teniendo en cuenta que **ordenado** tarda $O(n)$, entonces al final tendríamos $g = n \cdot 1$, por lo tanto $bogoMin \in O(n)$.

Ya en caso de un arreglo con propiedades de multiconjunto, el mejor caso sería tomar la primera permutación que esté ordenada, así que se cumple el mismo análisis.

Pregunta 2

Siendo N el número de personas en la fila F. Como se habla del costo amortizado para cada **llamado** de $sombrerear()$ se debe tener encuenta que c_i se define como el **costo real** de la i-ésima llamada. Este costo es igual al numero de acciones de poner/quitar el sombrero. Sabiendo que la acción de poner/quitar un sombrero es $O(1) y como el llamado se detiene en el momento de **poner** un sombrero, podemos decir que $c_i = m$ donde m es la cantidad de sombreros totales cambiados en el **llamado**.

Así, se define la función de transferencia $\Phi(F)$ asociado al estado de la fila F, como el número total de personas que tienen el sombrero **puesto**.

$$\Phi = \text{Número de personas con el sombrero puesto}$$

Entonces, el costo amortizado de cada llamado de $sombrerear()$ se define como

$$Amor_{c_i} = c_i + \Phi(F_i) - \Phi(F_{i-1})$$

Analizamos por casos

- Caso 1: Se tienen m cambios con $m < N$

  Esto es que $m-1$ personas en la fila se quitaron el sombrero y la k-ésima se lo colocó. De esto obtenemos que

  - Costo real: $c_i = m$ por haber m cambios.
  - Cambio potencial:
    * Estado $i-1$: Se quitan $m-1$ sombreros porque habian $m-1$ sombreros puestos.
    * Estado i: Se coloca 1 sombrero.

  Sustituyendo tenemos que

  $$Amor_{c_i} = m + 1 - (m-1) = 2$$

- Caso 2: Se tienen m cambios con $m = N$

  Esto es que todas las personas tenien el sombrero puesto y se lo quitaron.

  - Costo real: $c_i = m = N$ por haber $m = N$ cambios.
  - Cambio potencial:
    * Estado $i-1$: Se quitan $m = N$ sombreros porque habian $m = N$ sombreros puestos.
    * Estado i: 0 porque no se coloca ningun sombrero.

  Sustituyendo tenemos que

  $$Amor_{c_i} = N + 0 - N = 0$$

Notamos que

$$Amor_{c_i} = \begin{cases} 2 & \text{si la operación se detiene en algún punto (Caso 1)} \\ 0 & \text{si la operación recorre toda la fila (Caso 2)} \end{cases}$$

Dado que $2 \in O(1) \wedge 0 \in O(1)$, entonces orden amortizado para cada llamado de sombrerear() es $O(1)$.

Preguntа 3

> [!IMPORTANT] Estuve atacando el problema para probar $\oplus$-SAT $\in NP-Completo$ pero tras intentos fallidos, procedí a investigar en la internet y encontré un articulo de Wikipedia sobre XORSAR

que afirma $\oplus$-SAT $\in P$ y desde allí el enfoque de mi investigación cambio totalmente.

Para probar $\oplus$-SAT $\in P$ basta con el contrar un algoritmo que resuelva el problema en tiempo polinomial. Dicho algoritmo se describe a continuación:

Una fórmula de $\oplus$-SAT (XORSAT) tiene la forma:

$$(l_{1,1} \oplus l_{1,2} \oplus \dots) \wedge (l_{2,1} \oplus l_{2,2} \oplus \dots) \wedge \dots \wedge (l_{k,1} \oplus l_{k,2} \oplus \dots)$$

Para que la fórmula completa sea verdadera (satisfactible), todas las cláusulas deben ser verdaderas simultáneamente. La novedad es que **una cláusula individual C_j es verdadera si y solo si un número impar de sus literales son verdaderos**.

Esto es analizando cómo se comporta con múltiples operandos:

- Dos operandos: $P \oplus Q$ es verdadero si y solo si exactamente uno de P o Q es verdadero.
  - falso $\oplus$ falso = falso
  - falso $\oplus$ verdadero = verdadero
  - verdadero $\oplus$ falso = verdadero
  - verdadero $\oplus$ verdadero = falso
- Tres operandos: $P \oplus Q \oplus R$.
  - Si ninguno es verdadero:
    * falso $\oplus$ falso $\oplus$ falso = falso.
  - Si uno es verdadero:
    * verdadero $\oplus$ falso $\oplus$ falso = verdadero.
    * falso $\oplus$ verdadero $\oplus$ falso = verdadero.
    * falso $\oplus$ falso $\oplus$ verdadero = verdadero.
  - Si dos son verdaderos:
    * verdadero $\oplus$ verdadero $\oplus$ falso = falso.
    * verdadero $\oplus$ falso $\oplus$ verdadero = false.
    * false $\oplus$ verdadero $\oplus$ verdadero = false.
  - Si tres son verdaderos:
    * verdadero $\oplus$ verdadero $\oplus$ verdadero = verdadero.

Como podemos ver, el resultado es verdadero únicamente cuando el número de operandos verdaderos es impar (1 o 3 en este caso).

Podemos generalizar esta propiedad a una cláusula con cualquier cantidad de literales. Una cláusula en XORSAT, como $C_j = (l_1 \oplus l_2 \oplus \dots \oplus l_k)$, se evalúa aplicando el operador XOR de forma asociativa. Por ejemplo, $l_1 \oplus l_2 \oplus l_3$ es lo mismo que $(l_1 \oplus l_2) \oplus l_3$. El comportamiento clave de la cadena de operaciones XOR es que el resultado final es equivalente a sumar los valores booleanos (0 para falso, 1 para verdadero) y tomar el resultado módulo 2:

- Si la suma de los literales verdaderos es par, el resultado de la cláusula es falso ($0 \bmod 2 = 0$).

- Si la suma de los literales verdaderos es impar, el resultado de la cláusula es verdadero (1 mod 2 = 1).

Por lo tanto, para que una cláusula C_j sea verdadera, es una condición necesaria y suficiente que un número impar de sus literales sean verdaderos.

Sabiendo esto, **el primero paso clave es convertir la fórmula $\oplus$-SAT en un sistema de ecuaciones lineales sobre $\mathbb{Z}_2$.**

Paso 1: Convertir la fórmula XORSAT en un sistema de ecuaciones lineales

La traducción de la lógica booleana a la aritmética en $\mathbb{Z}_2$ (donde false=0 y true=1) es la siguiente:

- Cada literal l se convierte en una variable entera $x_l \in 0, 1$.
- La negación de un literal, $\neg l$, se traduce como $1 + x_l$.

Esto quiere decir, sea $T(l)$ la traducción de un literal l a $\mathbb{Z}_2$:

- Si l, entonces $T(l) = x_l$.
- Si $\neg l$, entonces $T(l) = 1 + x_l$.

Por lo tanto, la restricción de que la cláusula C_j debe ser verdadera se convierte en la ecuación lineal: $[T(l_{j,1}) + T(l_{j,2}) + \cdots + T(l_{j,p})] \pmod 2 \equiv 1$.

Dado que la fórmula completa de $\oplus$-SAT es una conjunción de k cláusulas $(C_1 \wedge C_2 \wedge \cdots \wedge C_k)$, para que sea satisfactible, todas las cláusulas deben ser verdaderas. Esto nos da un sistema de k ecuaciones lineales.

Por consiguiente, **el segundo paso clave es aplicar Eliminacion Gaussiana a este sistema de ecuaciones generado**.

Paso 2: Aplicar Eliminacion Gaussiana

El proceso es el siguiente:

- Construir la Matriz Aumentada: Se representa el sistema de k ecuaciones con n variables como una matriz aumentada $[A|b]$ de dimensiones $k \times (n+1)$.
  - n es el número de literales en la fórmula sin tomar en cuenta repeticiones o negaciones.
- Aplicar Eliminación Gaussiana: Se transforma la matriz a su forma escalonada mediante operaciones de fila elementales. En $\mathbb{Z}_2$, estas operaciones son la suma de una fila a otra (que es un XOR bit a bit) y el intercambio de filas.
- Determinar si existe solución: Una vez en forma escalonada, podemos determinar si el sistema tiene una, ninguna o múltiples soluciones. **Para el problema de satisfactibilidad, solo necesitamos saber si existe al menos una solución**.

Complejidad

Aca el artículo de Wikipedia sobre la Eficiencia Computacional de la Eliminacion Gaussiana.

- Para construir el sistema de ecuaciones se debe leer la fórmula completa. Si la longitud total de la fórmula es L, esto toma tiempo $O(L)$.
- El algoritmo de eliminación Gaussiana para una matriz de $k \times (n+1)$ se ejecuta en tiempo $O(max(k, n+1)^3)$. En el contexto de nuestro problema $\oplus$-SAT:
  - n es el número de literales en la fórmula sin tomar en cuenta repeticiones o negaciones.
  - k es el número de cláusulas.

Por lo que, en general el algoritmo completo en sí hace uso de dos algoritmos: Transformación de formula lógica a matriz y Eliminacion Gaussiana del cual tenemos tiempos polinomiales respectivamente, por consecuencia directa el algoritmo completo tendrá tiempo polinomial. Asi, hemos probado que $\oplus$-SAT $\in P$.

El debate de la cota superior para este algoritmo completo es interesante. * Tenemos un caso donde la traducción puede ser demorada y la eliminación Gaussiana constante. Si tenemos $formula = (p_1 \oplus p_2 \oplus ... \oplus p_m)$, esto es una formula donde tenemos una sola cláusula ($k = 1$), un literal p ($n = 1$) pero se repite m veces. Acá Eliminacion gaussiana será $O(1)$, sin embargo la traducción tardará $O(m)$ y podemos tener un m muy grande. * Por otro lado, sería intuitivo decir que $O(L)$ es acotada por $O(max(k, n+1)^3)$, pero no tiene sentido comparar directamente entre L, k y n dado que no existe relacion directa, por lo que será mejor especificar que $O(max(k, n+1)^3 + L)$.