José Alejandro Pérez Márquez 310109800
Tania Michelle Rubí Rojas 315121719
Erick Bernal Márquez 317042522

Tarea 3

Complejidad Computacional
Semestre 2023-1

Ejercicio 1

Leel el pdf adjunto, Algorithmic Problems and Their Complexity, y responder:

- Realizar un breve resumen, o diagrama ilustrativo, del texto. Debe contener, y resaltar, las respuestas a las siguientes preguntas:
  - ¿En qué consiste la tesis de Church-Turing? ¿Cuál es la diferencia con la versión extendida?
  - ¿Cuál es la importancia de la notación asintótica?
  - ¿Qué son los problemas de planificación (scheduling) y cuál es su importancia?
- Ejemplificar instancias de algunos de los problemas que se mencionan. Al menos 6 problemas diferentes (que no sean de variantes del mismo problema)
- Proponer una versión o variante de los problemas, del inciso anterior, que esté en P. Al menos dos deben ser problemas no triviales, y para estos se debe demostrar su pertenencia a la clase P.
- Mencionar algunos ejemplos de aplicaciones prácticas que tienen (o podrían tener) estos problemas.

- RESUMEN:

  El artículo *Algorithmic Problems and Their Complexity* está dividido en cuatro partes de las cuales explicaremos a continuación

  - **Problemas Algorítmicos**

    La primera sección está dedicada precisamente a dejar clara la definición de problema, sabemos que hay muchos tipos sin embargo nos interesan aquellos que se pueden resolver de manera lógica, que consiguen ser procesados por una computadora y sin ambigüedades, es decir aquellos que pueden ser resueltos por un algoritmo. A este tipo se les conoce como *Problemas algorítmicos*

    Un problema algorítmico está definido por:

    1. Una descripción del conjunto de entradas, los cuales pueden ser representados como una secuencia finita sobre un alfabeto finito
    2. Una descripción de una función que mapea cada entrada en un conjunto no vacío de salidas, los cuales cada uno puede también verse como una secuencia finita sobre un alfabeto finito

Por simplicidad los llamaremos *problemas*. Además, el autor nos da una breve clasificación de estos: problemas de búsqueda, de optimización, de evaluación o de decisión. Así mismo el tamaño y formato de entrada del alfabeto puede impactar fuertemente en el diseño y construcción del algoritmo, pues razonable pensar que si queremos ordenar 1000 números tomará mas tiempo que si queremos ordenar solo 10 números, independientemente de cuales sean estos, es por ello que nos concentramos particularmente en el tamaño de la entrada y no en la entrada en sí. Nos interesa más el proceso.

- **Algunos ejemplares importantes**

  En la segunda sección el autor presenta una lista de las familias de problemas más conocidos y nos explica brevemente en qué consisten cada uno de ellos. Algunos de ellos son:

  1. *Problema del agente viajero:* Consiste en visitar todas las ciudades una sola vez de manera que gastemos la menor cantidad de recursos y lleguemos al mismo lugar. Su importancia es más que obvia pues es una situación que se nos presenta cada vez que emprendemos un viaje. Desde hacer una gira por un continente, hasta visitar los diferentes museos de la CDMX.
  2. *Problemas de partición:* Ahora, si a la gira le añadimos que solo podemos transportar ciertos objetos que nos serán de más o menos utilidad se convierte en otro problema.
  3. *Problema de planificación:* Los problemas de planificación son una familia de problemas en los que en cada caso las tareas deben dividirse entre personas o máquinas sujetas a diferentes restricciones: No todas las personas son aptas para todas las tareas, diferentes personas o máquinas pueden tomar diferentes cantidades de tiempo para completar la misma tarea, ciertas tareas pueden necesitar completarse en un orden específico, puede haber horas de inicio más tempranas o más tardíos tiempos de finalización (plazos) especificados, y hay diferentes optimizaciones criterios que se pueden utilizar.
     Su importancia está en que resolver estos problemas minimizan la escasez de recursos para resolver las tareas y garantizan la equidad entre las partes que utilizan los recursos.
  4. *Problema de la galería de Arte:* Radica en determinar cual es el mínimo de cámaras que pueden estar dentro de un polígono tal que cada cámara vigile los demás lados.
  5. *Problema del campeonato:* Como su nombre nos puede dar a entender el problema consiste en determinar si un equipo de fútbol, baloncesto, tenis, etc, es capaz de ganar un campeonato dadas ciertas habilidades a cada jugador.

José Alejandro Pérez Márquez 310109800
Tania Michelle Rubí Rojas 315121719
Erick Bernal Márquez 317042522

Tarea 3

- **¿Cómo medimos el tiempo de cálculo de un algoritmo?**

  El autor aborda la cuestión de la complejidad en los algoritmos. Para la cual nos presenta una primera forma de hacerlo pero enseguida nos muestra que la definición resulta ser deficiente pues hay que tomar en cuenta varias cosas como lo es la computadora, el lenguaje, y la implementación del algoritmo. Es por ello que utilizamos un modelo de cómputo llamado Maquina de Turing (MT) donde solo importe el tamaño de la entrada y el algoritmo en sí.

  La tesis de Church-Turing nos da la seguridad para analizar este modelo y que los resultados sean equivalentes en otros modelos, pues afirma que que todos los modelos de computación pueden simularse entre sí, por lo que el conjunto de problemas solucionables algorítmicamente es independiente del modelo computacional. La versión extendida afirma que para cualesquiera dos modelos de computación R_1 y R_2 hay un polinomio p tal que t pasos de cálculo de R_1 en una entrada de longitud n pueden ser simulados por $p(t, n)$ pasos de cálculo de R_2. Es decir, existe una manera de cuantificar la pérdida o ganancia de eficiencia al pasar de un modelo a otro.

- **Complejidad de un problema**

  Finalmente, el autor nos habla de comparar la eficiencia de un algoritmo señalando que un algoritmo A es *al menos* tan rápido como A'. Sin embargo esto trae problemas ya que pueden haber casos donde el algoritmo A sea más eficiente para ciertas entradas que el algoritmo A' y viceversa.

  Por ello utilizamos cotas superiores o inferiores las cuales representamos mediante notación asintótica, esta notación nos permite compara la complejidad de los algoritmos y por lo tanto de problemas, por ello es importante porque nos permite clasificar problemas. Nos permite decir que un algoritmo A es *al menos tan rápido asintóticamente* como A'

  De esta manera el autor aborda la definición de complejidad algorítmica en términos del *worst-case runtime* (peor tiempo de ejecución) que es la medida del tiempo computacional más usada, con la cual se pasa a definir cómo comparar dos algoritmos para el mismo problema.

  Se dice que la complejidad algorítmica de un problema es $f(n)$ si el problema se puede resolver por medio de un algoritmo A que tenga como peor tiempo de ejecución de $O(f(n))$

- **Ejemplificar instancias de algunos de los problemas que se mencionan. Al menos 6 problemas diferentes (que no sean de variantes del mismo problema)**

1. Traveling salesperson problem: Un ejemplo concreto del TSP es el siguiente: Dado un conjunto de 5 ciudades, de las cuales se conoce para cada par de ellas la distancia que las separa, un agente viajero ha de partir de una ciudad de origen y debe visitar exactamente una vez cada ciudad del conjunto, y retornar al punto de partida. Un recorrido con estas características es llamado, dentro de este contexto, ciclo hamiltoniano o tour. El problema consiste en encontrar el ciclo para el cual la distancia total recorrida sea mínima.

2. Un ejemplo concreto de Scheduling problems es el VSP (vehicle scheduling problem) o problema de planificación vehicular en el cual hay un diversas de rutas para un conjunto de vehículos con capacidad limitada de carga, dichas rutas deben pasar por un número de ciudades o clientes para entregar la carga. El objetivo del VSP es realizar las entregas a un número de clientes asignando rutas de costo mínimo iniciando y terminando en el almacén o punto de partida.

3. Surveillance (or covering) problems: Dado un conjunto de elementos $U = \{1, 2, ..., m\}$ y n conjuntos cuya unión comprende el universo, el problema del conjunto de cobertura consiste en identificar el menor número de conjuntos cuya unión aun contiene todos los elementos del universo. Por ejemplo, sea $U = \{1, 2, 3, 4, 5\}$ y los conjuntos $S = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}$. Claramente, la unión de todos los conjuntos de S contiene todos los elementos de U. Sin embargo, podemos cubrir todos los elementos con el siguiente conjunto de elementos, con menor número de elementos: $\{\{1, 2, 3\}, \{4, 5\}\}$

4. Clique problem: Dada una red social de n personas y relaciones de amistad, se desea organizar una fiesta a la que asistan la mayor cantidad de personas todas amigas entre sí. Este problema claramente se puede modelar con un grafo donde los nodos son las personas y los ejes las relaciones de amistad entre las aristas. Aquí se desea conocer cuál es el subgrafo completo más grande (clique máxima) puesto que este subgrafo representará la mayor cantidad de personas amigas entre sí, y este conjunto de personas será la solución a nuestro problema.

5. Optimization problem: Un ejemplo de problema de optimización es el problema de Maximum weight matching, el cual consiste en encontrar, en una gráfica con pesos, un conjunto de aristas no adyacentes entre sí (matching) en el que se maximiza la suma de pesos.

6. Number theory problem: Un ejemplo de test de primalidad es la prueba de Lucas-Lehmer es una prueba que sirve para determinar si un determinado número de Mersenne M_p es primo.

- **Proponer una versión o variante de los problemas, del inciso anterior, que esté en P. Al menos dos deben ser problemas no triviales, y para estos se debe demostrar su pertenencia a la clase P.**

1. La variante de decisión del TSP. Es decir dado un recorrido sobre una gráfica que empiece y termine en el mismo vértice, verificar que dicho recorrido pase por todos los vértices únicamente una vez. Además todas las aristas tienen peso 1.

   Pertenencia a P:

   Para demostrar la pertenencia a P debemos dar un algoritmo polinomial determinista. El algoritmo es el siguiente.

   I. Marcar el primer vértice como visitado y pasar al siguiente indicado por el recorrido.
   II. Verificar el estado de visitado del siguiente vértice. Si está marcado como visitado entonces nuestra respuesta es un rotundo *no*. Si no lo está pasamos al siguiente y repetimos este mismo paso hasta llegar al último paso.
   III. Al llegar al ultimo vértice, verificamos que todos los vértices de la gráfica hayan sido visitados. Si lo fueron regresamos *Sí*, en caso contrario *No*.

   Este algoritmo toma cada vértice del recorrido y lo verifica, por lo que tiene que recorrer toda la cadena de entrada. Además en el último paso debemos verificar el estado de visita de todos los vértices. Así tiene una complejidad de $O(|V|)$

2. Variante de decisión de VSP: Proponemos una versión trivial donde solo exista un camión y pase por todas las ciudades con costo de 1. Esto se reduce a TSP visto anteriormente.

3. Surveillance (or covering) problems:

   Sea nuestro universo $U = \{1, 2, ...n\}$ de tamaño n y S un conjunto de subconjuntos de U como por ejemplo $S = \{\{1, 2, 3\}\{3, 4\}\{8, 9\}...\}$ ¿Cuál es el menor numero de unión de subconjuntos de S de tal manera que la unión sea U?

Pertenencia a P:

Para esto lo hacemos por medio de fuerza bruta, es decir.

I. Tomamos primero los conjuntos de S de tamaño n y verificamos que este sea U. Si lo es acabamos y si no seguimos con los demás conjuntos.

II. Una vez terminando de verificar los conjuntos de tamaño n verificamos que la unión de conjuntos de tamaño $n-1$ sea U. En caso de serlo ya hemos acabado, de lo contrario repetimos el paso con $n-2$, luego $n-3$ y así hasta llegar a conjuntos de tamaño 1.

III. Si ninguno de ellos resulta ser U entonces no hay solución

El algoritmo primero toma un conjunto de tamaño n, luego de tamaño $n-1$, luego $n-2$... esto nos da una primera idea de que tiene complejidad $O(n^2)$. Sin embargo dentro de los conjuntos de tamaño $n-1$ unimos primero dos conjuntos, luego 3, luego 4, así hasta al menos n. Lo mismo para conjuntos de tamaño $n-3$, $n-4$ hasta los conjuntos de tamaño 1 de los cuales puede haber en el peor caso n conjuntos. Podríamos hacer cálculos de combinatoria para saber exactamente cual es el número de pasos a ejecutar, sin embargo nos limitaremos a decir que está en $O(n^k)$ ya que solo nos interesa su pertenencia a P.

4. Clique problem para $k = 2$: donde k es el número de subgráficas que queremos encontrar.

I. Sea la gráfica G dada tomamos su complemento G' y trabajamos con este

II. Partimos de un vértice al azar de G' y lo coloremos de algún color c_1. Pasamos algún vértice vecino y verificamos el color. Si tiene color y resulta ser el mismo que el de su "padre" entonces no hay clique posible
En caso de que ya tenga color tenemos dos opciones:
   - El vértice "padre" tiene color c_1 por lo cual a su vecino le asignamos el color c_2
   - El vértice "padre" tiene color c_2 por lo cual a su vecino le asignamos el color c_1

III. Repetimos hasta que no haya vértices por colorear. Si la coloración fue exitosa entonces la coloración de los vértices nos darán los mismo vértices que hacen el clique

El algoritmo anterior recorre la gráfica y en el peor de los casos recorremos la gráfica completa, todos los vértices y todas las aristas por lo cual tiene complejidad $O(|V| + |E|)$.

5. Optimization problem: De manera similar al problema 3 (Surveillance problems), por fuerza bruta, es decir probamos todos los posibles casos.
   I. Construimos conjuntos de tamaño 1 con el peso de las aristas no adyacentes entre sí.
   II. Luego conjuntos de tamaño 2 con los pesos de aristas no adyacentes entre sí. Repetimos hasta que ya no queden más conjuntos por construir.
   III. Seleccionamos el conjunto que tenga mayor suma de los elementos (que es el peso de nuestras aristas) y tenga menor cantidad de elementos

   De igual manera primero tomamos conjuntos de tamaño 1, luego de tamaño 2, luego 3, esto nos da una $O(n^2)$. Así mismo podemos calcular las operaciones realizadas mediante combinatoria. Solo nos limitaremos a decir que está en $O(n^k)$
6. Number theory problem: Solo hace falta verificar el número con la prueba de Lucas-Lehmer el cual es
   I. Verificar si $S_{p-2} \equiv 0 \mod M_p$

   donde $S_i = \begin{cases} 1 \ si \ i = 0 \\ s_{i-1}^2 - 1 \ en \ otro \ caso \end{cases}$

   Cabe aclarar que si suponemos que estamos en un modelo de computo RAM entonces la complejidad es lineal, ya que lleva un numero constante de pasos pues tenemos operaciones elementales. Sin embargo podemos pasar el algoritmo a una MT del cual solo habrá una perdida de eficiencia (Tesis de Church-Turing extendida), en ambos modelos el tiempo de cálculo es un polinomio y por lo tanto está en P

- **Mencionar algunos ejemplos de aplicaciones prácticas que tienen (o podrían tener) estos problemas.**

  El problema del agente viajero sale de manera natural al ser una situación que se nos presenta regularmente en la vida diaria. Desde bandas o cantantes dando giras o tours por todo un continente o país hasta el caso en querer recorrer los museos de la CDMX. Resolver el problema de panera eficiente nos ahorra mucho dinero y tiempo al querer emprender cualquier tipo de viaje. Su aplicación práctica es per se.

  En el caso de que tengamos capacidad limitada para el viaje entonces se convierte en VSP, el cual también tiene aplicaciones per se.

  Ahora, si queremos acomodar nuestro equipaje en diferentes mochilas se convierte en un problema de cobertura. Pues responde a la pregunta ¿Cómo reparto el equipaje en todas las maletas de tal manera que ocupe menor cantidad de estas? No necesariamente iguales pues cada una tiene diferente capacidad.

El problema del clique puede tener aplicaciones en otro modelo llamado "Computo distribuido". Si queremos que un grupo de personas se ponga de acuerdo en una decisión entonces necesitamos encontrar una subgráfica completa y a partir de ahí podemos aplicar distintos protocolos para resolver este problema [1]

Una aplicación práctica de los números primos es la encriptación de mensajes al momento de generar llaves públicas y privadas.

Ejercicio 2

Sea $\Sigma = \{s_1, s_2, ..., s_k\}$, con $k \in \mathbb{N}$.

- **Demuestra que Σ^* es contable.**

  Para poder contar los elementos haremos una especie de mapeo entre los números naturales y las cadenas en Σ^*. Así pues, asignamos un número natural a las cadena de longitud 1, luego a las cadenas de longitud, después a las cadenas de longitud 3, así sucesivamente. Es decir

  $$1 \to 0$$
  $$2 \to 1$$

  Luego cadenas de longitud 2.

  $$3 \to 00$$
  $$4 \to 01$$
  $$5 \to 10$$
  $$6 \to 11$$

  Después longitud 3.

  $$7 \to 000$$
  $$8 \to 001$$
  $$9 \to 010$$
  $$10 \to 011$$
  $$\dots$$

  Y así consecutivamente para la longitud n de las cadenas. En el caso de la cadena vacía ϵ le asignamos el 0

  De esta manera demostramos que Σ^* es contable.

[1] Problema del consenso: Todos los nodos de una gráfica deben tener la misma salida.

- **¿El conjunto de máquinas de Turing NO-Deterministas es contable?**

  Sabemos cómo codificar una MT-determinista (MTD) sin embargo ¿Cómo codificamos una MT-No determinista (MTND)? La respuesta es que la podemos codificar de la misma manera que una MTD. Y podemos contar ese conjunto de MTND de la misma manera que contamos Σ^* es decir, haciendo un mapeo entre cada número natural y las MTND. Por lo tanto decimos que el conjunto de MTND es contable.

- **¿Qué podemos concluir de las afirmaciones de los incisos anteriores?**

  Si tomamos en cuenta que Σ^* es una manera de representar las MT y por lo tanto contarlas, podemos pensar lo mismo con respecto a las MTND. Es decir ambas son contables, y de manera similar podemos hacer una biyección entre ambos conjuntos lo cual nos indica que nuestros conjuntos son equinumeros, es decir hay el mismo número de Maquinas de Turing deterministas que Maquinas de Turing no deterministas.

Referencias

- Texto adjunto:

  `https://drive.google.com/file/d/1YiPMw23NoP-fUtIVuVwtwG01oKM9XEtY/view`

- Problema del agente viajero:

  `https://gc.scalahed.com/recursos/files/r161r/w25199w/M1CCT07B_Caso2_S2.pdf`

- VSP: `https://www.sciencedirect.com/science/article/abs/pii/S0925527308003514`

- Problema del clique: Práctica 1 del presente curso.

- Prueba de Lucas-Lehmer:

  `https://numerosprimos.org/prueba-de-lucas-lehmer/`

- Numeros primos en la criptografia:

  `https://www.logaritmoneperiano.com/numeros-primos-y-criptografia/`

- Codificación de Maquinas de Turing. Visto en clase.

- Maquinas de turing contables `https://cs.stackexchange.com/questions/18099/is-the-set-of-non-deterministic-turing-machines-countable`