Tarea 3 \Complejidad computacional

Victor Hugo Gallegos Mota
316160456

José Demian Jiménez
314291707

Luis Alberto Hernández Aguilar
314208682

October 13, 2022

1 Ejercicio 1

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

- Realizar un breve resumen, o diagrama ilustrativo, del texto. Debe contener, y resaltar, las respuesta a las siguientes preguntas:
  - ¿En que consiste la tesis de Church-Turing? ¿Cual es la diferencia con la version 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)

  Por ejemplo para los problemas que impliquen operaciones con teoria de numeros vendrian siendo los problemas que impliquen el encriptado y desencriptado de mensajes ya que muchos de estos sistemas usan operaciones de modulo para poder llevar a cabo un encriptado y a la vez un desencriptado de algun mensaje como por ejemplo el sistema de cifrado vigenere.

  El problema de la mochila (KP) tiene amplias aplicaciones en diferentes campos, como la programación de máquinas, la asignación de espacios y

la optimización de activos. Mientras tanto, es un problema difícil debido a su complejidad computacional, pero se han desarrollado numerosos enfoques de solución para una variedad de KP.

Problema del viajante de comercio (TSP):

Dado un conjunto de ciudades y la distancia entre cada par de ciudades, el problema es encontrar la ruta más corta posible que visite cada ciudad exactamente una vez y regrese al punto de partida. Tenga en cuenta la diferencia entre el ciclo hamiltoniano y TSP.

El problema TSP suele aparecer en algunos otros problemas como un subproblema en algunos otros campos como por ejemplo el de la secenciacion de ADN.

Partitioning problems bin packing problem:

Problema dinámico conocido como problema de embalaje en contenedores en línea, aquí los elementos llegan en momentos arbitrarios uno por uno. esta secuencia de artículos debe empaquetarse en contenedores de tamaño unitario, de modo que se minimice el número máximo de contenedores utilizados durante todo el tiempo.

League Championship Algorithm (LCA):

Es un algoritmo basado en población estocástica propuesto recientemente para la optimización global continua que intenta imitar un entorno de campeonato en el que los equipos artificiales juegan en una liga artificial durante varias semanas (iteraciones). Dado el calendario de la liga en cada semana, una cantidad de individuos como equipos deportivos juegan en parejas y el resultado de su juego se determina en términos de victoria o derrota (o empate), dada la fuerza de juego (valor de condición física) junto con la formación del equipo previsto/ arreglo (solución) desarrollado por cada equipo.

- 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.

Para los problemas que impliquen cliques tenemos el siguiente , la identificacion del clique maximo. Un método codicioso básico puede encontrar una sola camarilla más grande. Hacer crecer la camarilla actual un vértice a la vez recorriendo los vértices restantes del gráfico, comenzando con una camarilla arbitraria(por ejemplo , cualquier vértice individual o incluso el conjunto completo). Para cada vértice examinado por este ciclo, agregue v a la camarilla si está cerca de cada vértice actualmente en la camarilla, de lo contrario, descarte v. Este algoritmo es lineal en el tiempo. Debido a la simplicidad con la que se pueden encontrar las camarillas máximas,

así como a su posible tamaño modesto.

Los problemas de mochila anidados pueden ocurrir como subproblemas de un tipo importante del problema del material de corte donde hay etapas de corte sucesivas. En estos problemas solo se permite dividir (y no reagrupar) los elementos de la mochila entre cada etapa.

El Problema del Viajante con su variante Ventanas de Tiempo (TSPTW) es una de las variantes del problema del TSP más estudiadas y conocidas. Aplicado a un viajante de comercio, consiste en buscar la ruta de coste mínimo que debe realizar éste último, empezando y volviendo a un mismo y único almacén, pasando por todos los nodos una sola vez en la franja horaria especificada por el cliente, conocida como ventana de tiempo

Una variante del algoritmo TSP viene siendo el TSP-Asimetrico el cual consiste en que la distancia entre dos nodos en red del TSP es la misma en ambas direcciones. El caso donde la distancia de A a B no es igual que la distancia de B a A es llamado asimétrico.

TSPN : las distancias vienen de 1,...,N;

En esta variante del problema de embalaje en contenedores, cada artículo debe asignarse a un contenedor, sin el conocimiento de los artículos posteriores. además, no se permite la migración de artículos de un contenedor a otro, no se pueden mover artículos.

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

  Algunos problemas que impliquen el uso del problema de la mochila son aplicables a los problemas de embalaje en contenedores, para ayudar a resolver este problema.

  El TSP surge naturalmente como un subproblema en muchas aplicaciones de transporte y logística, por ejemplo, el problema de organizar rutas de autobuses escolares para recoger a los niños en un distrito escolar.

  El problema del camarilla (clique) como aplicación, se mostrará que un problema clásico de programación de una sola máquina se puede reducir a este problema de camarilla trabajando con un gráfico de conflicto de trabajo subyacente. También se avanzan algunas técnicas de preprocesamiento (o kernelización) para reducir el tamaño del gráfico.

Resumen:

La noción de “problema”, tal como se la utiliza comúnmente, es tan general que ser imposible de formalizar. Para restringirnos a algo más manejable, consideraremos solo ”problemas algorítmicos”. Por un algoritmo problema, nos referimos a un problema que es adecuado para el procesamiento por computadoras y para el cual el conjunto de resultados correctos es inequívoco. El problema de encontrar una sentencia justa para un acusado no es algorítmico, ya que depende de cuestiones de filosofía judicial y, por lo tanto, no es adecuado para su procesamiento por computadoras.

La tesis clásica de Church-Turing dice que todos los modelos de computación pueden simularse entre sí, de modo que el conjunto de problemas solucionables algorítmicamente es independiente del modelo de computación, en cambio La tesis extendida de Church -Turing va un paso más allá:

Para cualquiera de los dos modelos de cálculo R1 y R2 hay un polinomio p tal que t pasos de cálculo de R1 en una entrada de longitud n pueden ser simulado por p(t, n) pasos de cálculo de R2.

La dependencia de p(t, n) de n solo es necesaria en el caso de que un algoritmo (como la búsqueda binaria, por ejemplo) se ejecute durante un tiempo sublineal. Por supuesto, no es justo considerar que todas las operaciones aritméticas son pasos de cálculo igualmente costosos. Consideramos que la división (hasta un número fijo de lugares decimales) consume más tiempo que la suma. Además, el tiempo realmente requerido depende de la longitud de los números involucrados. Si descendemos al nivel de bits, entonces cada operación aritmética sobre números de longitud de bit l requiere al menos $\Omega(l)$ operaciones de bits.

La tesis extendida de Church-Turing debe ser revisada a la luz de los nuevos tipos de computadoras. No hay duda de que es correcto en el contexto de las computadoras digitales . Incluso las llamadas computadoras de ADN ”solo” dan como resultado chips más pequeños o un mayor grado de paralelismo. Esto puede representar un enorme avance en la práctica, pero no tiene efecto sobre el número de operaciones elementales.

Dejamos que tA(x) denote el tiempo de cálculo del algoritmo A en la entrada x en el modelo de costo unitario para un modelo de cálculo elegido (por ejemplo, máquinas registradoras). Ahora podemos intentar comparar dos algoritmos A y A para el mismo problema de la siguiente manera: A es al menos tan rápido como A si $tA(x) \leq tA\ (x)$ para todo x. Esta definición obvia es problemática por varias razones:

- El valor exacto de tA(x) y por lo tanto la comparación de A y A depende del modelo de cálculo.

- Solo para algoritmos muy simples podemos esperar calcular tA(x) para

todo x y probar la relación tA(x) $\leq$ tA (x) para todo x.

- A menudo, cuando comparamos un algoritmo A simple con un algoritmo A que es más complicado pero bien adaptado al problema en cuestión, encontramos que tA(x) ¡ tA (x) para entradas "pequeñas" x pero tA(x) ¿ tA (x) para "grande" entradas x.

La medida más utilizada para el tiempo de cálculo es el tiempo de ejecución en el peor de los casos:

tA(n) := $sup\{tA(x) :\leq n\}$

Frecuentemente tA(n) := $sup\{tA(x) := n\}$ y tA = tA cuando t A es monótonamente creciente. Este es el caso de la mayoría de los algoritmos. El uso de tA(n) asegura que el tiempo de ejecución en el peor de los casos sea siempre una función monótonamente creciente, y esto será útil más adelante.

Finalmente podemos decir que la complejidad algorítmica de un problema es f(n) si el problema se puede resolver por medio de un algoritmo A con un tiempo de ejecución en el peor de los casos de O(f(n)), y cada algoritmo para este problema tiene un tiempo de ejecución en el peor de los casos de (f(n)). En este caso, el algoritmo A tiene el tiempo de ejecución asintóticamente mínimo. Sin embargo, no definimos la complejidad algorítmica de un problema porque los problemas no necesariamente tienen un tiempo de ejecución asintóticamente mínimo.

2 Ejercicio 2

Sea Σ=$\{s_1, s_2, ..., s_n\}$ con k $\in$N

- Demuestra que Σ^* es contable.
  Sea $c_s = |s| + \sum_{x \in s} x \mid s \in \Sigma$ la *clase* de s.
  Observemos que $\forall k \in N$ existe un número finito de cadenas en Σ^* de clase k, pues, para que una cadena s tenga clase k, la cardinalidad de s debe ser menor o igual a k y cada elemento de s debe ser menor a k. Entonces, a lo más existen k^k cadenas de clase k, pues no podemos tener más de una cadena distinta de longitud k con clase k.
  Con este hecho podemos construir una función inyectiva que asigne a todas las m cadenas de clase 0 los primeros m naturales comenzando con el 0, luego a las siguientes n cadenas de clase 1 los siguientes n naturales y así sucesivamente.
  Por lo tanto, como podemos mapear Σ^* al conjunto de los números naturales, concluimos que Σ^* es contable.

- ¿El conjunto de máquinas de Turing NO-Deterministas es contable?
  Sí, pues podemos codificar cada máquina de Turing no determinista como

una cadena finita de 0's y 1's. Esto haría que el conjunto de máquinas de Turing no deterministas sea como el del inciso anterior, el cual ya sabemos que es contable, y por eso es que sí es contable.

- ¿Qué podemos concluir de las afirmaciones de los incisos anteriores?
  Que para saber si un conjunto es contable basta con buscar una represetación del conjunto que se asemeje a Σ^* y, si existe, entonces sabremos que el conjunto es contable.

3 Referencias

- https://shareok.org/handle/11244/319274#:~:text=Knapsack%20problem%20(KP)%20has%20broad,for%20a%20variety%20of%20KP.
- https://drive.google.com/file/d/1YiPMw23NoP-fUtIVuVwtwG01oKM9XEtY/view
- https://www.math.uwaterloo.ca/tsp/apps/index.html
- https://www.sciencedirect.com/science/article/abs/pii/0377221793E0211F
- https://link.springer.com/article/10.1007/s43069-021-00111-x
- https://www.nayuki.io/page/countable-sets-and-kleene-star
- https://classroom.google.com/u/1/w/NTM4MDcxMDgzNjEz/t/all
- https://www.cs.rpi.edu/~goldberg/14-CC/02-ndt.pdf
- https://es.wikipedia.org/wiki/Problema_del_viajante#TSP_m%C3%A9trico
- https://cs.stackexchange.com/questions/18099/is-the-set-of-non-deterministic-turing machines-countable