.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1

Fecha de entrega: 24/11/2023

- Gauler, Ian Benjamin, Padrón: 109437
- Jauregui, Melina Belén, Padrón: 109524
- Tourne Passarino, Patricio, Padrón: 108725

Índice

1. Introducción

Este informe tiene como propósito analizar el Hitting Set Problem y explorar diversas soluciones que proponemos.

1.1. Información de la problemática

El enunciado propuesto, nos da la siguiente información:

- Scaloni tiene a su disposición el conjunto A de $n = 43$ jugadores $a_1, a_2, \ldots, a_{43}$.
- Existen m medios cada uno con un grupo de jugadoes favorito $B_1, B_2, \ldots, B_m$.
- Scaloni necesita conocer el grupo C más pequeño de jugadores de tal forma que cada medio B_i tenga al menos un jugador en el equipo.

/sectionAnálisis de la complejidad del problema

El problema de Scaloni es un caso particular del Hitting-Set Problem, cuya definición formal es: Dado un conjunto A de n elementos y m subconjuntos $B_1, B_2, \ldots, B_m$ de A ($B_i \subseteq A \forall i \in \mathbb{N}_m$), buscamos $C \subseteq A / \forall j \in \mathbb{N}_m, C \cap B_j \neq \emptyset$.

Hitting-Set Problem también tiene una versión de desición: Dado un conjunto A de n elementos y m subconjuntos $B_1, B_2, \ldots, B_m$ de A ($B_i \subseteq A \forall i$), y un número k, ¿existe un $C \subseteq A / |C| \leq k \wedge \forall j \in \mathbb{N}_m, C \cap B_j \neq \emptyset$.

Hitting-Set Problem se encuentra en NP porque se puede verificar en tiempo polinomial. Dado un subconjunto C, basta con verificar que $|C| \leq k$ y verificar que para cada conjunto B_j, C contiene algún elemento del mismo. La primera operación es de complejidad $O(n)$ porque basta con contar los elementos de C. La segunda es de complejidad $O(n \times m)$ porque para cada conjunto B_j ($j \in \mathbb{N}_m$) ($O(m)$) se debe recorrer C ($O(n)$) y verificar que cada elemento pertenezca a B_j ($O(1)$ si B_j se implementa como set).

Además, Hitting-Set Problem es un problema NP-Completo. Para demostrarlo basta con reducir polinomialmente un problema NP-Completo a Hitting-Set. Para esto, elegimos Dominating Set que a su vez es NP-Completo porque se puede reducir Vertex Cover al mismo, que demostramos en clase que es NP-Completo.

Recordemos que el problema de Dominating Set: Dado un grafo G, su busca un conjunto de vértices C para todo vértice $v \in G$, este está contenido en C o existe un vértice en C adyacente de v.

La reducción Dominating-Set $\leq_p$ Hitting-Set consta en lo siguiente: Dado un grafo G de n vértices del problema Dominating Set, Para cada vértice v_i, se construye un grupo B_i con el mismo y todos sus vértices adyacentes. Entonces, el grupo resultado del Hitting Set con los subconjuntos $B_1, B_2, \ldots, B_m$ será también el grupo resultado del Dominating Set. *Nota: Para cada k-clique dentro del grafo habrá hasta k sets iguales.* La complejidad de esta reducción depende de la implementación del grafo: Si los vértices desconocen a sus adyacentes, es $O(n^2)$ porque para cada vértice v_i se debe recorrer todos los vértices de G para verificar si son adyacentes a v_i; en cambio, si los vértices tiene referencia a sus adyacentes, la complejidad es $O(n \times o)$, siendo o el promedio del grado entre vértices, que puede variar entre 0 y m. En ambos casos, se trata de una complejidad polinomial.

$$\text{Dominating-Set} \leq_p \text{Hitting-Set}$$

La reducción Vertex-Cover $\leq_p$ Dominating-Set consta en lo siguiente: Dado del grafo G con n vértices y m aristas del problema Vertex Cover, para cada par de vértices adyacentes $v - w$, se agregan ambos al grafo G' (del problema Dominating Set) junto a su arista y se agrega un tercer vértice auxiliar vw, adyacente a los otros dos. A será el conjunto de vértices auxiliares. Luego,

del conjunto V' de k vértices solución de Dominating Set, se debe agregar a V (solución de Vertex Cover) todos los vértices de V' que no estén en el conjunto de auxiliares y, para cada vértice en $V' \cap A$ se agrega a V cualquiera de sus adyacentes. La primera parte de la reducción es $O(m)$ y la segunda es $O(k), k \leq n$, por lo que la complejidad total es $O(m + k)$, que es polinomial.

$$\text{Vertex-Cover} \leq_p \text{Dominating-Set}$$

Finalmente:

$$\begin{matrix}\text{Vertex-Cover} \leq_p \text{Dominating-Set} \\ \text{Dominating-Set} \leq_p \text{Hitting-Set}\end{matrix} \quad \overset{\text{por transitividad}}{\Longrightarrow} \quad \text{Vertex-Cover} \leq_p \text{Hitting-Set}$$

$$\Longrightarrow \text{Hitting-Set} \in \text{NP-Completo}$$

1.2. Cota de complejidad temporal

El algoritmo presentado tiene una complejidad temporal de $\mathrm{O}(n \log n)$. Esto se debe porque para obtener la solución óptima, sólo requerimos ordenar los compilados de mayor a menor tiempo de análisis respecto a los ayudantes. Para ello utilizamos sorted de la librería de python, que usa por detrás el algoritmo de Timsort, teniendo este una complejidad de $\mathrm{O}(n \log n)$.

2. Mediciones

Para las mediciones, creamos listas de compilados de ejemplo de forma aleatoria. Decidimos que el tiempo de análisis de un compilado fuera un valor aleatorio entre 1 y 99'999, basándonos en los datos de ejemplo provistos por la cátedra.

Medimos el tiempo de ejecución de nuestro algoritmo para ciertas cantidades de compilados.

Notamos que los procesos de fondo del sistema nos generaba distorciones en los tiempos de ejecución de una misma muestra. Atacamos este problema realizando varias mediciones para el mismo escenario de compilados y tomando el promedio de los tiempos obtenidos.

Además, como definimos cada escenario arbitrariamente, en una cantidad de compilados x, este escenario puede ser poco favorable (puede estar muy desordenado), mientras que en una cantidad de compilados $x + 1$ puede ser muy favorable. Por lo cual la diferencia de tiempos va a ser muy grande a pesar de que la cantidad de compilados solamente difiere en una unidad (TimSort es una combinación de inserción y mergeSort, por lo cual el desorden inicial toma gran relevancia en el rendimiento final del algorítmo). Para solucionar esto, decidimos hacer varios escenarios distintos para la misma cantidad de compilados.

Con el objetivo de comparar los tiempos de ejecución de nuestro algoritmo con la complejidad teórica, optamos por realizar tanto un análisis de regresión lineal como uno de regresión lineal logarítmica que se ajustara a nuestros datos. Para evaluar qué curva se ajusta mejor, usamos la raíz del error cuadrático medio (RMSE). Realizamos este análisis en un intervalo con tamaños pequeños (hasta 1'000) y en un intervalo con tamaños grandes (hasta 10'000). Ver figuras **??** y **??**

Se puede observar que en valores pequeños, el tiempo de ejecución del algoritmo sigue una clara curva lineal logarítmica. Sin embargo, a medida que aumenta el tamaño de la muestra de compilados, su comportamiento se aproxima más a una complejidad lineal.

Esto ocurre debido a la naturaleza de la función lineal logarítmica, que presenta una pendiente logarítmica. Cuando se trabajan con valores grandes, esta pendiente se vuelve casi constante, lo que la asemeja a una función lineal. Esto se debe a que el logaritmo es una función monótona creciente, lo que significa que su derivada es siempre positiva para valores mayores que cero. Sin embargo, el crecimiento de esta función se vuelve cada vez más lento a medida que los valores

aumentan, indicando que su segunda derivada es siempre negativa. En consecuencia, para valores muy grandes, la función lineal logarítmica se estabiliza y, aunque no es completamente constante, su comportamiento se asemeja a una función lineal.

De esta manera, mediante el gráfico y la validación a través del error cuadrático medio, hemos comprobado que la curva lineal logarítmica se adapta de manera más precisa a nuestros datos.

De esta forma pudimos comprobar empíricamente que la complejidad tiende a $O(n \log n)$.

Nótese que graficamos también dos curvas que demarcan una estimación de los cuantiles verticales 0,1 y 0,9. Esta estimación se realizó calculando los cuantiles para un grupo pequeño centrado en cada punto. Estas curvas nos ayudan a dimencionar cómo la varianza del tiempo de ordenamiento crece con el aumento del tamaño de la información de entrada.

Adicionalmente, graficamos la densidad de los tiempos que también brinda una visualización de cómo aumenta la varianza con el aumento del tamaño de los datos de entrada. Ver figuras **??** y **??**

3. Aproximación

3.1. Algoritmo Greedy

Proponemos dos aproximaciones extra. La primera es un algoritmo greedy que calcula la cantidad de grupos

4. Conclusiones

Finalmente, consideramos que la solución óptima para abordar la problemática de Scaloni sería que éste analizara los compilados en función del tiempo requerido por los asistentes para visualizar cada uno, organizándolos en orden descendente. Esta estrategia permitiría resolver el problema con una complejidad algorítmica de orden $O(n \log n)$. Este enfoque garantiza la máxima eficiencia en la visualización de los compilados y permite que el tiempo invertido por Scaloni y sus asistentes se administre de manera óptima.