Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2283 - Diseño y Análisis de Algoritmos
Profesor: Nicolás Van Sint Jan
Ayudante: Dante Pinto

Ayudantía 13

Teoría de Números y Complejidad Promedio

Problema 1: Grupos Conmutativos

Un grupo $(G, \circ)$ se dice conmutativo si para todos $x, y \in G$ se cumple que $x \circ y = y \circ x$. Como notación, definimos $[a, b] = a^{-1} \circ b^{-1} \circ a \circ b$.

1. Demuestre que $a \circ b = b \circ a$ si y solo si $[a, b] = 1$.

2. Decimos que un grupo es generado por $S \subseteq G$ si todo elemento $g \in G$ se puede expresar como producto de elementos e inversos de elementos en S.

   Desarrolle y analice un algoritmo que dado un conjunto finito $S = (g_1, ..., g_n)$ y una operación binaria $\circ$, determine si el grupo generado S y $\circ$ es conmutativo.

3. Definimos el centro de un grupo G como:

   $$Z(G) = \{x \in G \mid \text{ para todo } g \in G. [x, g] = 1\}$$

   Además para cada $g \in G$, definimos el centralizador de g como:

   $$C(g) = \{x \in G \mid [x, g] = 1\}$$

   Demuestre que $Z(G)$ es un subgrupo de G y que para todo $g \in G$, $C(g)$ es un subgrupo de G.

4. Dado un grupo $G = \langle g_1, ..., g_n \rangle$, definimos un subproducto aleatorio como:

   $$r = g_1^{\varepsilon_1} \circ ... \circ g_n^{\varepsilon_n} \in G$$

   donde cada ε_i se elige de forma uniforme e independiente del conjunto $\{0, 1\}$.

   Demuestre que si H es un subgrupo propio de G, entonces:

   $$Pr[r \notin H] \geq \frac{1}{2}$$

5. Desarrolle y analice un algoritmo aleatorizado que dados generadores $S = g_1, ..., g_n$ y una operación binaria $\circ$ determine si el grupo generado por S y $\circ$ es conmutativo.

Problema 2: Ordenamiento en tiempo lineal

Suponga que tiene una lista de largo n con números enteros, elegidos de forma uniforme e independiente del intervalo $[0, 100]$.

1. Diseñe un algoritmo que ordene la lista en tiempo lineal.

   Solución: Sabemos de cursos anteriores que ningún algoritmo basado en comparaciones puede ordenar en tiempo mejor que $\mathcal{O}(n \log(n))$, por lo que debemos diseñar un algoritmo que no se base en comparaciones. Uno de estos algoritmos es el *Counting Sort*. Una versión de este algoritmo es:

```
1  Function CountingSort(L)
2  |  c = [0 for i in range(101)]
3  |  ret = []
4  |  foreach i ∈ L do
5  |  |  c[i]++
6  |  idx = 0 foreach j ∈ c do
7  |  |  for i = 0; j; i++ do
8  |  |  |  ret.append(idx)
9  |  |  |  idx++
10 |  return ret
```

2. Analice la complejidad de su algoritmo en el peor caso.

   Solución: En el peor caso, el primer for del algoritmo tomará tiempo $\mathcal{O}(n)$, el segundo for tomará tiempo constante, $\mathcal{O}(101)$, y, si bien el tercer for está anidad con el segundo, considerando que cada j corresponde a uno de los contadores de números, al sumarlos todos tendremos tiempo lineal $\mathcal{O}(n)$. Por lo anterior, el tiempo total del algoritmo será lineal.

3. Analice la complejidad de su algoritmo en el caso promedio.

   Solución: Podemos ver que la cantidad de pasos no depende del input, solo depende de su tamaño, por tanto el caso promedio será el mismo que el mejor y/o peor caso, es decir, tomará tiempo lineal.

4. ¿Cómo se compara su algoritmo con el algoritmo de Quicksort?

   Solución: El algoritmo es completamente distinto al algoritmo de Quicksort, sin embargo no necesariamente es mejor, pues asume que se están ordenando números entero y además hay una dependencia oculta del rango de estos en el $\mathcal{O}(101)$.

Problema 3: Déjà vu

Suponga que tiene una lista de largo n con números reales, elegidos de forma uniforme e independiente del intervalo $[0, 100]$.

1. Diseñe un algoritmo que ordene la lista en tiempo lineal.

   Solución: Para hacer esto podemos utilizar BucketSort:

```
1  Function BucketSort(k)
2  |  buckets = [[] for i in range(k)]
3  |  ret = []
4  |  foreach i ∈ L do
5  |  |  idx =⌊k · i/m⌋
6  |  |  buckets[idx] = i
7  |  foreach b ∈ buckets do
8  |  |  b.InsertionSort()
9  |  return concat(buckets)
```

2. Analice la complejidad de su algoritmo en el peor caso.

   Solución: Dado que el primer for toma tiempo lineal, el peor caso para el algoritmo vendrá dado por el segundo for y ocurrirá cuando todos los elementos queden en el mismo bucket. En este caso la complejidad se la llevará el `InsertionSort`, es decir, tomará tiempo cuadrático.

3. Analice la complejidad de su algoritmo en el caso promedio.

   Solución: La complejidad del algoritmo vendrá dada por el segundo for, y estará en:

$$\mathcal{O}\left(E\left[\sum_{i=1}^{k}|b_i|^2\right]\right)$$

   Donde b_i es cada uno de los buckets. Luego, podemos definir la variable aleatoria:

$$X_{ij} = \begin{cases} 1 & \text{si } L[i] \in b_i \\ 0 & \text{en otro caso} \end{cases}$$

$$\Rightarrow b_i = \sum_{j=1}^{n} X_{ij}$$

$$\begin{aligned} E[b_i^2] &= E\left[\sum_{j=1}^{n}\sum_{l=1}^{n} X_{ij}X_{il}\right] \\ &= E\left[\sum_{j=1}^{n} X_{ij}^2\right] + E\left[\sum_{j=1}^{n}\sum_{l=1.l\neq j}^{n} X_{ij}X_{il}\right] \\ &= \sum_{j=1}^{n} E\left[X_{ij}^2\right] + \sum_{j=1}^{n}\sum_{l=1.l\neq j}^{n} E\left[X_{ij}X_{il}\right] \end{aligned}$$

   Calculando las esperanzas tenemos:

$$E\left[X_{ij}^2\right] = 1^2 \cdot \frac{1}{k}$$

$$E\left[X_{ij}X_{il}\right] = 1^2 \cdot \frac{1}{k} \cdot \frac{1}{k} = \frac{1}{k^2}$$

   Por tanto:

$$\begin{aligned} E[b_i^2] &= \sum_{j=1}^{n} E\left[X_{ij}^2\right] + \sum_{j=1}^{n}\sum_{l=1.l\neq j}^{n} E\left[X_{ij}X_{il}\right] \\ &= \sum_{j=1}^{n} \frac{1}{k} + \sum_{j=1}^{n}\sum_{l=1.l\neq j}^{n} \frac{1}{k^2} \\ &= n \cdot \frac{1}{k} + n(n-1)\frac{1}{k^2} \end{aligned}$$

   Finalmente, suponiendo que k es lineal respecto a n, tenemos que

$$\mathcal{O}(E[b_i^2]) = \mathcal{O}(n \cdot \frac{1}{n} + n(n-1)\frac{1}{n^2}) = \mathcal{O}(n)$$

4. ¿Cómo se compara su algoritmo con el algoritmo de Quicksort?

   Solución: Nuevamente el algoritmo parece no tener relación con quick sort, pues no se basa en comparaciones, sin embargo si consideramos 2 buckets y un quick sort que siempre elige el pivote en el medio de la lista, ambos algoritmos tienen un comportamiento similar.