IIC2283 — Diseño y Análisis de Algoritmos
Ayudantía 2

Juan Navarro

Problema 1

1. Se considerará que existe un algoritmo que permite multiplicar dos polinomios de grado i en tiempo $O(i \log i)$, y también que es posible multiplicar un polinomio de grado 1 por otro de grado i en tiempo $O(i)$.

   Describa un algoritmo que dado un conjunto $\mathcal{C}$ de j valores enteros, encuentre un polinomio $p \in \mathbb{Z}[x]$ de grado j, tal que el coeficiente de x^j sea 1, y que $p(c) = 0$ para todo elemento $c \in \mathcal{C}$. Analice el tiempo de ejecución del algoritmo.

   Idea de solución El polinomio p corresponde a $(x - c_1)(x - c_2) \cdots (x - c_j)$, por lo que el problema corresponde a calcular el producto de j polinomios de grado 1. Existen dos estrategias principales: uno por uno (por ejemplo, de izquierda a derecha), con tiempo $O(n^2)$ dado por que $T(1) = O(1)$ y $T(n) = O(n) + T(n-1)$, o resolviendo recursivamente cada mitad, y luego multiplicando los resultados, con tiempo dado por $T(1) = O(1)$ y $T(n) = 2T\left(\frac{n}{2}\right) + O\left(\frac{n}{2} \log \frac{n}{2}\right)$, que resulta en $O(n \log^2 n)$.

2. Describa una recurrencia donde C_n corresponda a la cantidad de palabras de largo $2n$, sobre el alfabeto $\{(,)\}$, donde los paréntesis se encuentren correctamente balanceados.

   Idea de solución Toda palabra correctamente balanceada comienza con un paréntesis abierto, para el cual existe un paréntesis cerrado correspondiente. Por lo anterior, toda palabra del lenguaje puede describirse como:

   $$(w_1)w_2,$$

   donde w_1 y w_2 están balanceadas. Luego, se consideran las posibilidades para el largo de w_1. Si w_1 tiene largo $2m$, entonces w_2 tiene largo $2n - 2m - 2$, y por lo tanto existen $C_m \cdot C_{n-m-1}$ palabras posibles en ese caso.

   Finalmente, considerando todos los largos posibles para w_1, entre 0 y $2(n-1)$, se obtiene que:

   $$C_n = \sum_{i=0}^{n-1} C_i \cdot C_{n-i-1},$$

   donde $C_0 = 1$, por que la palabra vacía se considera balanceada.

Problema 2

Demuestre que encontrar el k-ésimo elemento de un conjunto no ordenado, de n elementos distintos, requiere al menos $n + k - 2$ comparaciones cuando $k \leq \frac{n+1}{2}$.

Para esto considere el siguiente juego entre un algoritmo y un adversario:

- El input son n valores desconocidos distintos.
- En cada paso el algoritmo pide comparar dos valores, y el adversario retorna el resultado de la comparación. En esta fase, el adversario puede modificar libremente los valores involucrados, siempre que no cambie las relaciones de orden ya establecidas.
- Termina cuando el algoritmo determina el k-ésimo elemento.

Parte 1 Justifique que una cota en la cantidad de comparaciones en este juego corresponde a una cota en el problema original.

Idea de solución Al ser un algoritmo basado en comparaciones, los valores de los elementos no son importantes, ya que solo se utilizan las relaciones de orden entre elementos.

Más formalmente, si existe un algoritmo para encontrar el k-ésimo elemento entre n elementos, que realiza a lo más ℓ comparaciones con cualquier input, lo podemos utilizar para ganar el juego dado con a lo más ℓ comparaciones, independientemente de la estrategia del adversario. Esto por que los elementos a comparar los elige el algoritmo, y el adversario no puede modificar las relaciones de orden que ya fueron definidas.

Por lo anterior, si demostramos que el adversario tiene una estrategia con la cual el algoritmo debe realizar al menos m comparaciones, también demostramos que todo algoritmo para el problema original debe superar esa cota para algún input.

Parte 2 Muestre que el adversario tiene una estrategia que obliga al algoritmo a realizar al menos $n+k-2$ comparaciones, cuando $k \leq \frac{n+1}{2}$.

Idea de solución La estrategia del adversario se basa en mantener 3 conjuntos: L, G, U, donde U son los elementos indeterminados, L son los elementos menores a todos los elementos de U, y G son los elementos mayores a todos los elementos de U.

Los conjuntos L y G se mantienen ordenados, y se construyen a partir de U. Para esto, en algunos casos se traspasa un elemento de U a L o G, asignándolo como el nuevo elemento máximo o mínimo del conjunto respectivo.

Además, el adversario mantiene que cada elemento de U está comparado con a lo más 1 elemento del mismo conjunto. De esa forma, U está formado por elementos aislados, y pares ordenados aislados entre sí.

Por la estrategia todas las comparaciones quedan definidas, a menos que se comparen dos elementos de U. Si ambos elementos están aislados, entonces el adversario elige uno como menor que el otro, y los mantiene en U, y de lo contrario, existen 5 casos posibles:

- Comparar dos elementos maximales,
- Comparar dos elementos minimales,
- Comparar un elemento maximal con uno minimal,
- Comparar un elemento maximal con uno aislado, y
- Comparar un elemento minimal con uno aislado.

Se denotará por $M(n, k)$ la cantidad de comparaciones para encontrar el elemento k de un conjunto de n elementos.

En cada uno de los casos, el adversario puede modificar el valor maximal o minimal involucrado en la comparación, de forma de que sea el máximo o mínimo dentro de U, y así asignarlo a L o G. En el primer caso, si el tamaño de U es igual a n, y se comparan dos elementos maximales, entonces la cantidad de comparaciones puede acotarse por:

$$M(n, k) \geq 2 + M(n-1, k),$$

por que al eliminar un elemento de U, el problema se transforma a encontrar el k-ésimo elemento entre los $n-1$ restantes, y además se pierden dos comparaciones: la que generó el par ordenado inicial, y la del último paso.

En los cinco casos se puede determinar una recurrencia similar, y luego demostrar por inducción que $M(n,k) \geq n+k-2$.

Problema 3

Se considera la siguiente versión del problema de la mochila:

- El input consiste de n objetos y la capacidad de la mochila. Cada objeto tiene asociado un peso y un valor, que se asumen positivos.
- Se busca el subconjunto de objetos de mayor valor, tal que su peso total no exceda la capacidad de la mochila. Si existe más de un subconjunto de valor máximo, se prefiere el de menor peso.
- Subconjuntos distintos tienen distinto peso o distinto valor.
- No se pueden repetir ni fraccionar objetos: es decir es el problema 0/1.

Se denotará por $w(x)$ al peso de la solución x, y por $val(x)$ a su valor, donde x se interpreta como un vector en $\{0,1\}^n$ donde cada posición indica si el objeto correspondiente pertenece o no al subconjunto.

Luego, se define que una solución x domina a otra solución y, si se cumple que:

$$w(x) \leq w(y) \text{ y } val(x) \geq val(y),$$

con lo cual se define que una solución x es dominadora, si no existe otra solución y tal que y domine a x.

Parte 1 Demuestre que la solución óptima al problema es una solución dominadora.

Idea de solución Por contradicción.

Como notación, se definirá la operación $z = x + y$ de forma que $z_j = 1$ si y solo si $x_j = 1$ o $y_j = 1$. Además, se define la solución 1_j como el subconjunto que solo contiene al elemento j.

Considere el siguiente algoritmo, donde c representa la capacidad de la mochila:

1. Se define $S_0 = \{\vec{0}\}$.
2. Para $j = 1, \ldots, n$, sea $T_j = \{x + 1_j \mid x \in S_{j-1}\}$, y
$$S_j = S_{j-1} \cup T_j - \{y \in S_{j-1} \cup T_j \mid y \text{ es dominado por algún } x \in S_{j-1} \cup T_j\}$$
3. Retornar la solución $x^* \in S_0 \cup \cdots \cup S_n$ de mayor valor tal que $w(x^*) \leq c$, y si existe más de una solución de valor máximo, la de menor peso.

En adelante se utilizará q_j para denotar una cota superior al tamaño de S_j.

Parte 2 Demuestre que el conjunto S_j puede calcularse en tiempo $O(q_j)$. Para esto describa un primer algoritmo de tiempo $O(q_j \log q_j)$, y luego modifique la representación de los datos.

Idea de solución El primer algoritmo se obtiene ordenando el conjunto por peso no decreciente, y valor decreciente cuando los pesos son iguales. Luego, basta revisar el valor de las soluciones en orden, eliminado los elementos con valor menor al último no eliminado.

Como los conjuntos se construyen inductivamente, se pueden mantener como listas ordenadas. Luego se puede obtener T_j como una lista ordenada en tiempo lineal, simplemente agregando el elemento j a los elementos de S_{j-1}, y luego mezclarla (como en mergesort) con S_{j-1}, obteniendo el conjunto S_j ordenado. Se continúa como en el caso anterior.

Parte 3 Asumiendo que las cotas q_j se eligen de forma que $q_1 \leq q_2 \leq \cdots \leq q_n$, demuestre que el algoritmo funciona en tiempo $O(nq_n)$.

Idea de solución Si las cotas son no decrecientes, entonces q_n tiene valor máximo. Cada conjunto S se calcula en tiempo $O(q_n)$, y existen n conjuntos, por lo que se construyen en tiempo $O(nq_n)$.

Luego, si se ordenan los $m = 2^n$ subconjuntos posibles por peso no decreciente, y valor decreciente cuando los pesos son iguales, se obtienen las soluciones: $x_1, \ldots, x_m$. Con ellos, se define $A_0 = \{\}$ y $A_k = \{x_1, \ldots, x_k\}$, para k entre 1 y m, y luego δ_k como:

$$\delta_k = (\max_{x \in A_k} val(x)) - (\max_{x \in A_{k-1}} val(x)).$$

Parte 4 Demuestre que x_k es una solución dominadora si y solo si $\delta_k > 0$, y además que $\sum_{\ell=1}^{k} \delta_\ell = val(x_k)$.

Idea de solución δ_k siempre es mayor o igual a 0, por que $A_k \supset A_{k-1}$. Si δ_k es cero, entonces existe una solución anterior con valor mayor o igual, y peso menor o igual. Por otra parte, si $\delta_k > 0$, entonces ninguna solución anterior la domina, por que el valor es mayor, y tampoco es dominada por otra solución posterior, por que tienen mayor peso, o menor valor.

Si se asume que los valores de los objetos son aleatorios, con distribución uniforme(0,1), entonces δ_k corresponderá a la variable aleatoria Δ_k. De forma similar, la cota q_i corresponderá a la variable aleatoria Q_i.

Parte 5 Determine el valor de $\mathbb{E}(val(x_m))$

Solución $n/2$. La solución x_m contiene todos los objetos, y cada uno tiene valor esperado $1/2$.

Parte 6 Suponiendo que $\mathbb{E}[\Delta_k | \Delta_k > 0] \geq \frac{1}{32n^2}$, demuestre que:

$$\sum_{k=1}^{m} Pr[\Delta_k > 0] \leq 16n^3$$

Idea de solución Desarrollar la expresión:

$$\frac{n}{2} = \mathbb{E}[val(x_m)] = \mathbb{E}\left[\sum_{i=1}^{m} \Delta_i\right]$$

Parte 7 Muestre que en promedio, el algoritmo pertenece a $O(n^4)$.

Idea de solución Se define una variable aleatoria D_i, tal que $D_i = 0$ si $\Delta_i = 0$ y $D_i = 1$ si $\Delta_i > 0$. Luego, se tiene que

$$\mathbb{E}[Q_n] = \mathbb{E}\left[\sum_{i=1}^{m} D_i\right] \leq 16n^3,$$

y por lo tanto que en promedio el algoritmo pertenece a $O(n^4)$.