Examen 3

Análisis y Diseño de Algoritmos

21 de febrero de 2022

Duración: 2 horas
No se permite ningún tipo de apunte

Ejercicio 1 (2 ptos). Muestre la tabla para el problema Subset-sum visto en clase para los items $[1, 2, 2, 1]$ con $W = 4$. Solo es necesario que muestre la tabla resultante final.

Ejercicio 2 (3 ptos). Sobre el problema de subsecuencia creciente máxima visto en clase.

- Demuestre la siguiente propiedad: si $X = (i_1, i_2, \ldots, i_{k-1}, i_k)$ es una subsecuencia creciente máxima dentro de las que terminan en i_k, entonces $X' = (i_1, i_2, \ldots, i_{k-1})$ es una subsecuencia creciente máxima dentro de las que terminan en i_{k-1}.
- Escriba la recurrencia vista en clase para $OPT(i)$, que guarda el tamaño de una subsecuencia creciente máxima que termina en i

Ejercicio 3 (4 ptos). Dado un arreglo $A[1..n]$ de números enteros, decimos que un subconjunto de índices de $\{1..n\}$ es **independiente** si no existen dos índices cuya diferencia es exactamente 1. Por ejemplo, si $n = 5$, un subconjunto independiente es $\{1, 3, 5\}$, pero $\{1, 2, 4\}$ no lo es. Queremos diseñar un algoritmo de programación dinámica que encuentra un subconjunto de índices independiente cuya suma de elementos correspondientes es máxima. Por ejemplo, si $A = [2, 6, 5, 2]$, la respuesta sería 8, correspondiente al subconjunto de índices $\{2, 4\}$.

- Sea $OPT(i)$ el valor de una solución óptima para el arreglo $A[1..i]$. Describa una recurrencia para $OPT(i)$ (no necesita probar su correctitud).
- Diseñe un algoritmo de programación dinámica (haga el pseudocódigo) con tiempo de ejecución $O(n)$ a partir de la recurrencia anterior.

Ejercicio 4 (2 ptos). Indique la salida para el problema de mochila fraccionaria luego de ejecutar el algoritmo visto en clase con la siguiente entrada: $v = [10, 22, 10, 40, 90, 56]$, $w = [1, 2, 5, 8, 10, 7], W = 16$.

Ejercicio 5 (4 ptos). Dados dos conjuntos A y B de números enteros positivos, un *emparejamiento* entre A y B es un conjunto de pares ordenados $\{(a,b) : a \in A, b \in B\}$, tales que todo elemento en A aparece exactamente una vez, y todo elemento en B aparece exactamente una vez. El *costo* de un emparejamiento X es $\sum_{(a,b)\in X} ab$. Por ejemplo, si $A = \{3,2,5\}$, $B = \{1,6,4\}$, un emparejamiento podría ser $\{(3,1),(2,6),(5,4)\}$ y su costo es $3 \cdot 1 + 2 \cdot 6 + 5 \cdot 4 = 35$.

Diseñe un algoritmo voraz que encuentra un emparejamiento de costo mínimo.

(a) Mencione cual es la elección voraz

(b) Escriba el pseudocódigo (no código) de su algoritmo voraz. Indique claramente qué recibe y qué devuelve su algoritmo. El algoritmo debe ser recursivo.

(c) Demuestre que su elección voraz es correcta. Enuncie a modo de lema y demuestre que su lema es correcto.

Deberá utilizar las siguientes variables para soluciones devueltas por el algoritmo o utilizadas en las demostraciones: X, X'. Para subproblemas: A', B'.

Ejercicio 6 (2 ptos). Ilustre la operación del algoritmo Counting-sort en el arreglo $A = [6,0,2,0,1,3,4,6,1,3,2]$. Deberá mostrar los resultados parciales de los arreglos involucrados iteración por iteración.

Ejercicio 7 (2 ptos). Indique como ordenar n números enterios en el rango de 0 a $n^3 - 1$ en $O(n)$ **usando radix sort**. Basta dar una idea bien definida y un ejemplo concreto.

Ejercicio 8 (2 ptos). Ilustre la operación del algoritmo Bucket-sort en el arreglo $A = [0{,}79; 0{,}13; 0{,}16; 0{,}64; 0{,}39; 0{,}20; 0{,}89; 0{,}53; 0{,}71; 0{,}42]$.