

Estructuras de Datos y Algoritmos – IIC2133

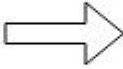
I2

9 de octubre, 2018

1. Backtracking

Un tablero de *kakuro square* es una grilla de $N \times M$ donde cada posición tiene una celda que puede ser rellenada por un número natural distinto de 0. Además, cada fila y columna tiene un número natural que indica el valor que debe tener la suma de los números correspondientes a la fila o columna. P.ej.:

	23	15	12
23			
17			
10			



	23	15	12
23	9	8	6
17	8	4	5
10	6	3	1

El problema de rellenar la grilla cumpliendo con las restricciones se puede resolver utilizando la estrategia de backtracking.

- Identifica las variables del problema y sus respectivos dominios.
- Explica con tus palabras cómo se podría revisar en tiempo $O(1)$ si la asignación de una variable rompe una restricción.
- Explica con tus palabras una poda o una heurística que se pueda aplicar para resolver este problema más eficientemente.

Respuesta:

- a) Para resolver con backtracking este problema se debe asignar a cada celda un número, por lo que las variables son las celdas [1pto].

En teoría cada celda puede tener cualquier número natural, pero ya que no puede haber un dominio infinito para resolverlo con backtracking es necesario acotar los dominios. Un dominio acotado correcto puede ser los números del 1 al $\text{MAX}(\text{restriction})$. Ya que se sabe que nunca un número puede ser mayor al valor de su restricción [1pto].

- b) Se puede mantener un contador por cada restricción del problema que cuente la suma actual de cada fila o columna. De esta manera se inmediatamente si rompo la restricción cuando el contador supera la restricción [2pts].

- c) [2pts]

Podas: Ver que los contadores descritos en b) no superen el máximo antes de tener completa la fila o columna. Ver que la suma de una fila o columna actual más el número de casillas vacías de la fila o

columna no sea mayor a la restricción (ya que no se pueden asignar ceros). Probablemente hay mas podas.

Heurísticas: Probar primero las celdas que pertenecen a filas o columnas con más valores ya asignados (o menos valores por asignar). También pueden haber muchas más.

2a) Hashing

Queremos multiplicar dos números X y Y , y para verificar si el resultado $Z = X \times Y$ es correcto, aplicamos una función de hash h a los tres números y vemos si

$$h(h(X) \times h(Y)) = h(Z).$$

Si los números difieren, entonces cometimos un error, pero si son iguales, entonces el resultado es (muy probablemente) correcto. h se define como calcular repetidamente la suma de los dígitos, hasta que quede un solo dígito, en cuyo caso 9 cuenta como 0.

P.ej., si $X = 123456$ y $Y = 98765432$, entonces $Z = 12193185172992$, y $h(X) = h(21) = 3$, $h(Y) = h(44) = 8$, $h(Z) = h(60) = 6$, y efectivamente, $h(3 \times 8) = h(24) = 6 = h(Z)$. Notemos que el dígito 9 no influye en el resultado calculado de h ; p.ej., $h(49) = h(13) = 4$.

a) Da una expresión matemática para $h(X)$ [**1.5 pts.**]; y **b)** muestra que este método de verificación del resultado de la multiplicación es correcto [**1.5 pts.**].

Respuesta:

- a) **X mod 9** (También se aceptan respuestas en donde el alumno calcula la suma de los dígitos con alguna expresión y después puso módulo 9, o expresiones recursivas, funciones compuesta y otras donde se mencione explícitamente que $h(9) = 0$ o se maneje bien ese caso)
- b)

Sabemos que:

$$X = 9 \times Q_1 + R_1$$

$$Y = 9 \times Q_2 + R_2$$

$$X \times Y = Z$$

Y queremos demostrar que:

$$h(h(X) \times h(Y)) = h(Z)$$

$$h(h(9 \times Q_1 + R_1) \times h(9 \times Q_2 + R_2)) = h(X \times Y)$$

$$h(h(9 \times Q_1 + R_1) \times h(9 \times Q_2 + R_2)) = h((9 \times Q_1 + R_1) \times (9 \times Q_2 + R_2))$$

En la expresión de la izquierda podemos eliminar todos los términos múltiplos de 9 ya que estamos trabajando en mod 9, obteniendo

$$h(R_1 \times R_2) = h((9 \times Q_1 + R_1) \times (9 \times Q_2 + R_2))$$

$$h(R_1 \times R_2) = h(81 \times Q_1 \times Q_2 + 9 \times Q_1 \times R_2 + 9 \times Q_2 \times R_1 + R_1 \times R_2)$$

Nuevamente, eliminamos los términos múltiplos de 9 de la expresión de la derecha ya que estamos trabajando en mod 9, obteniendo

$$h(R_1 \times R_2) = h(9 \times (9 \times Q_1 \times Q_2 + Q_1 \times R_2 + Q_2 \times R_1) + R_1 \times R_2)$$

$$h(R_1 \times R_2) = h(R_1 \times R_2)$$

Notas: Se aceptaran demostraciones menos formales, incluso con palabras, siempre que mencionen que los cocientes se eliminan ya que se está trabajando en mod 9 y quedan los restos. No se aceptarán respuestas que solo mencionan la propiedad sin demostrarla. Si el alumno no uso **X mod 9** en la parte a, se deberá evaluar caso a caso.

2b) Tablas de hash

Para cada uno de los siguientes problemas, responde si es posible resolver el problema eficientemente mediante *hashing*. En caso afirmativo, explica claramente cómo; de lo contrario, sugiere otra forma de resolverlo según lo estudiado en el curso.

- a) Considera un sistema de respaldo en que toda la información digital de una empresa tiene que ser respaldada, es decir, copiada, cada cierto tiempo. Una propiedad de estos sistemas es que solo una pequeña fracción de toda la información cambia entre un respaldo y el siguiente. Por lo tanto, en cada respaldo, solo es necesario copiar la información que efectivamente ha cambiado. El desafío es, por supuesto, encontrar lo más que se pueda de la información que no ha cambiado. [1.5 pts.]

Si es posible resolver mediante hashing. Un ejemplo de implementación eficiente mediante hashing es el uso de una función de hash que inicialmente se haya usado para respaldar toda la información. Estos elementos habrían llegado a algún espacio de la tabla. Al momento de querer respaldar nuevamente la información que cambió, uno puede tomar cada archivo de la información digital y utilizar el hash para identificar si este se modificó en la tabla (por ejemplo, hashear el nombre del archivo con su path y la fecha de modificación, o también hashear el archivo completo) y en caso de que coincidan, no se respalda. En caso de que sean diferentes, se puede generar el nuevo hash a partir de un hash incremental y los pocos cambios generados para luego insertar nuevamente en la tabla (liberando la anterior o complementando con el resto de la información nueva a respaldar).

- b) Dada una lista L de números, queremos encontrar el elemento de L que sea el más cercano a un número dado x . [0.5 pts.]

No es resoluble por hashing eficientemente. Dado que es una lista sin un orden específico, se debe considerar alguna alternativa que entregue un orden para luego hacer por ejemplo una búsqueda binaria sobre un arreglo o una búsqueda sobre un árbol binario balanceado.

- c) Queremos encontrar un string S de largo m en un texto T de largo n . [1 pt.]

Se vio en clases. Se puede resolver eficientemente mediante hashing. Usando una función de hash incremental, se toman los primeros m caracteres del texto de largo n para calcular el hash. De esta manera se compara con el hash del string S a buscar. En caso de que no sean iguales los hash, debes eliminar el primer elemento del string y agregar el siguiente del texto ($O(1)$) para luego volver a realizar el proceso. En caso de que existan colisiones, es mejor revisar caracter a caracter en caso de que sean iguales los hash.

3. Rotaciones + árboles de búsqueda balanceados

- a) [Teorema fundamental de las rotaciones]. Muestra que cualquier árbol binario de búsqueda (no necesariamente balanceado) puede ser transformado en cualquier otro árbol binario de búsqueda con las mismas claves mediante una secuencia de rotaciones simples.
- b) Determina un orden en que hay que insertar las claves 1, 3, 5, 8, 13, 18, 19 y 24 en un árbol 2-3 inicialmente vacío para que el resultado sea un árbol de altura 1, es decir, una raíz y sus hijos.
- c) Considera un árbol rojo-negro formado mediante la inserción de n nodos usando el procedimiento visto en clase. Justifica que si $n > 1$, el árbol tiene al menos un nodo rojo.
- d) Muestra cómo construir un árbol rojo-negro que demuestre que, en el peor caso, casi todas las rutas desde la raíz a una hoja tienen largo $2 \log N$, en que N es el número de nodos del árbol.

Respuesta:

a) Demostrar recursivamente. Dados dos árboles A y B con las mismas claves en distinto orden (respetando las propiedades de los ABB), una posibilidad es tomar el nodo raíz de A y buscarlo en B, y luego hacer las rotaciones necesarias para llevarlo a la raíz, y luego repetir el proceso recursivamente sobre los subárboles siguientes.

Alternativamente, se puede mostrar que todo ABB se puede reordenar mediante rotaciones simples para llevarlo a una lista ligada (visto como un árbol en que todos los nodos tienen solo hijos derechos o izquierdos), la cual sería igual para todo ABB con las mismas claves, y dichas rotaciones se pueden deshacer, llegando así de cualquier árbol A a un árbol B con las mismas claves.

b) Hay varios órdenes posibles. Para que el árbol tenga altura 1 este tiene que quedar con 5 y 18 en la raíz y con 1 y 3 en el hijo izquierdo, 8 y 13 en el medio y 19 y 24 en el derecho. Un orden posible es 1-18-24-5-8-13-19, pero en todo caso es necesario mostrar la construcción del árbol y como va quedando paso a paso hasta llegar al resultado indicado.

c) Se puede demostrar inductivamente. Comenzando por el caso base, deben explicar las reglas de construcción de árboles rojo-negro (específicamente que los nodos insertados siempre son rojos), y luego explicar el paso inductivo mediante los casos posibles para la inserción de un nodo (padre negro, padre rojo con tío rojo, padre rojo con tío negro) de manera de mostrar que siempre queda al menos un nodo rojo después de que la inserción es balanceada.

d) Partiendo del caso en que un árbol rojo-negro fuese compuesto solo de nodos negros (esto es, un ABB normal balanceado), se tiene que la altura de este es $O(\log n)$ para n nodos. Luego, teniendo en cuenta que todo nodo rojo tiene hijos negros, el peor caso en términos de altura es aquel árbol en que se alternan nodos rojos y negros a cada nivel, de manera que su altura es del doble que el árbol mencionado previamente pero mantiene aún las propiedades de los árboles rojo-negros.

Alternativamente, se puede mencionar la altura de los árboles 2-4 (que siempre son transformables a un árbol rojo-negro y viceversa), siendo esta $O(\log n)$, y mostrar cómo esta aumenta al ser transformada en un árbol rojo negro, con un ejemplo que muestre un árbol 2-4 cuya altura aumente al doble al ser transformado en rojo-negro.