Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación
Universidad de Chile
CC3101-01 Matemáticas de Discoteca para la Computación

Profesor: Pablo Barceló
Auxiliares: Tomás Martínez, Ignacio Riego

Auxiliar 5

Jueves 4 de Mayo, 2017

P1. Sea Σ un alfabeto finito y Σ^* el conjunto de las palabras finitas formadas or símbolos en Σ. Dada una palabra $w = a_1...a_n \in \Sigma^*$, se define su palabra *reversa* por $w^R = a_n...a_1$.

a) Dé una definición recursiva del operador potencia $w^i = w \dots w$, donde w está i veces, e $i \in N$

Solución:

$w^0 = \varepsilon$ donde ε es el string vacío

$w^i = ww^{i-1}$

b) Muestre que $\forall i \in N, w \in \Sigma^*$ se cumple que $l(w^i) = i \cdot l(w)$, donde $l(w)$ es el largo de w.

Solución:

Primero recordamos la definición de $l(w)$

$l(\varepsilon) = 0$

$l(aw) = 1 + l(w)$

Con esto podemos demostrar que $l(wz) = l(w) + l(z)$:

Si $w = a_i a2...a_k$

$l(wz) = 1 + l(a_2...a_k z)$ Después de k pasos

$l(wz) = k + l(z)$ y sabemos que $l(w) = k$, entonces queda demostrado.

Ahora $l(w^i) = l(w)l(w^{i-1})$. Si seguimos esta iteración obtenemos $l(w) = il(w) + l(w^0)$ pero $l(w^0) = l(\varepsilon) = 0$

Con esto conclíumos $l(w^i) = il(w)$

c) Muestre que $\forall i \in N, w \in \Sigma^*$ se cumple que $(w^i)^R = (w^R)^i$

Solución:

Recordamos que w^R se define como:

$\varepsilon^R = \varepsilon$

$(wa)^R = a(w^R)$

Es propiedad conocida que si $w, z \in \Sigma^*, (wz)^R = z^R w^R$. Usamos esto sobre la definición de w^i:

$(w^i)^R = (ww^{i-1})^R = (w^{i-1})^R (w^R)^1$

Podemos seguir iterando y en el paso k:

$(w^{i-k})^R (w^R)^k = (ww^{i-k-1})^R (w^R)^k = (w^{i-k-1})^R w^R (w^R)^k$

Usando la definición de w^i:

$(w^{i-k-1})^R w^R (w^R)^k = (w^{i-k-1})^R (w^R)^{k+1}$

Hasta que eventualmente en el paso i:

$(w^0)^R (w^R)^i = (w^R)^i$

P2. Asuma que tenemos n tareas $t_1, ..., t_n$, y a con cada tarea t'_i $(1 \leq i \leq n)$ asociamos la siguiente información:

- El momento $c_i \geq 0$ en que la tarea comienza y otro $f_i > c_i$ en que finaliza.
- El beneficio $b_i > 0$ de que la tarea se ejecute.

Decimos que las tareas t_i y t_j son compatibles si sus intervalos de ejecución no se intersectan; es decir, si $f_i < c_j$ o $f_j < c_i$. Asuma ahora que las tareas $t_1, ..., t_n$ se hallan ordenadas no decrecientemente con respecto a los términos de finalización de tareas; es decir, para todo $1 \leq i \leq j \leq n$ se cumple que $f_i \leq f_j$.

Defina $B : 0, ..., n \to N$ de tal forma que $B(i)$ corresponde al mayor beneficio que se puede obtener al ejecutar algunas de las i primeras tareas de forma compatible. Formalmente:

$$B(i) = max\{\sum_{t_j \in A} b_j \mid A \subseteq \{t_1, ..., t_i\} \;\; t.q. \;\; \text{no hay dos tareas distintas en A que sean incompatibles.}\}$$

Defina recursivamente a B(i) en términos de $B(i-1)$ y $B(pred(i))$, donde $pred(i)$ es el mayor $j <$ i tal que t_j y t_i son compatibles.

Solución:

B(i) es la respuesta al problema si solo consideramos las primeras i tareas.

Si conocemos $B(i)$, para conocer $B(i+1)$ basta con determinar si vamos a realizar la tarea t_{i+1} o no.

Para determinar esto tenemos que comparar el mejor resultado sin realizar t_{i+1} y el mejor resultado realizándola.

Por definición el mejor resultado sin realizarla es $B(i)$, el mejor resultado usando solo las tareas anteriores.

Ahora, el mejor resultado realizándola es: el beneficio de realizar t_{i+1}, osea b_{i+1}) sumado a la mejor solución tomando solo las tareas que terminan antes de c_{i+1}, osea que terminan antes que t_{i+1} empieze. Esto es porque si decidimos realizar t_{i+1} no podemos realizar ninguna tarea que ocurra al mismo tiempo que t_{i+1}.

En otras palabras:

$B(i+1) = max(B(i), B(pred(i+1)) + b_{i+1})$

Porque $pred(i+1)$ representa la tarea quetermina más tarde pero que es compatible con t_{i+1}.

Ojo que si todas las tareas anteriores a t_{i+1} termiana antes que c_{i+1} puede darse que $pred(i+1) = i$ pero eso no afecta al algoritmo ya que tomamos máximo.

P3. Definimos el arbol Stern-Brocot de la siguiente forma:

Casos base:

Definimos la raiz del arbol r como un nodo que tiene un valor asociado $\frac{1}{1}$. Esta no tiene padres y tiene un hijo derecho y un hijo izquierdo.

Caso inductivo:

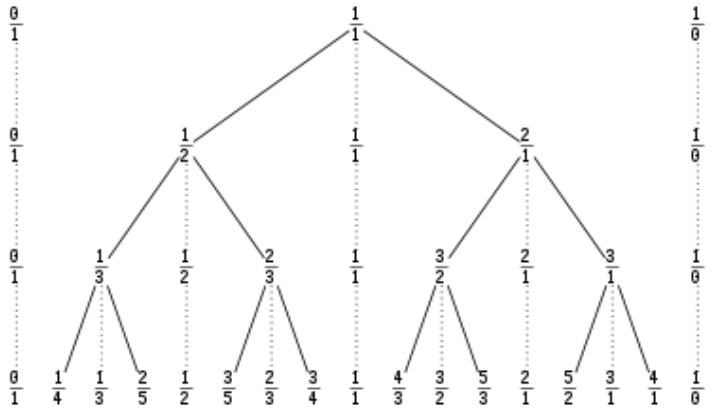
Dado un nodo a_p con valor asociado $\frac{m_p}{n_p}$ este tiene un hijo izquierdo y uno derecho.

Si un nodo a_{hd} es un hijo derecho de a_p y a_D es el ancestro más cercano de a_{hd} que está a su derecha. El valor asociado a a_{hd} es $\frac{m_p+m_D}{n_p+n_D}$.

Si no hay ningún ancestro a la derecha se toma $m_D = 1, n_D = 0$

De forma similar un nodo a_{hi} es un hijo izquierdo de a_p y a_I es el ancestro más cercano de a_h que está a su izquierda. El valor asociado a a_{hi} es $\frac{m_p+m_I}{n_p+n_I}$.

Si no hay ningún ancestro a la derecha se toma $m_I = 1, n_I = 0$



Diseñe un algortimo que dada una fracción $\frac{a}{b}$ irreductible encuentre el nodo dentro del arbol que la tiene como valor asociado. Puede asumir que toda fracción está en el arbol y que todas las fracciones del árbol son irreductibles, sin demostrarlo.

hint: piense en un árbol binario de búsqueda

Solución:

Vamos a demostrar que este árbol se comporta como un arbol binario de búsqueda, para esto primero probaremos:

si $\frac{m_1}{n_1} \leq \frac{m_2}{n_2}$ entonces $\frac{m_1}{n_1} \leq \frac{m_1+m_2}{n_1+n_2} \leq \frac{m_2}{n_2}$

Para esto usaremos $m_1 n_2 \leq m_2 n_1$:

$m_1 n_1 + m_1 n_2 \leq m_1 n_1 + m_2 n_1$

Factorizando:

$m_1(n_1 + n_2) \leq n_1(m_1 + m_2)$

Y multiplicando:

$\frac{m_1}{n_1} \leq \frac{m_1+m_2}{n_1+n_2}$

La otra desigualdad se hace igual:

$m_1 n_2 \leq m_2 n_1$

Sumo $m_2 n_2$ a ambos lados

$m_1 n_2 + m_2 n_2 \leq m_2 n_1 m_2 n_2$

factorizando y dividiendo:

$\frac{m_1+m_2}{n_1+n_2} \leq \frac{m_2}{n_2}$

Ahora que tenemos esta propiedad falta ver que todo nodo es menor que su hijo derecho y mayor que su hijo izquierdo.$(*)$

Para mostrar eso primero mostraremos que dado un nodo a_p, el nodo a_D su primer ancestro que se ecuentra a su derecha tiene una fracción asociada que es mayor que la asociada a a_p. Luego mostraremos lo mismo para a_I pero con menor en vez de mayor. De esto deduciremos $(*)$.

Caso Base: La raiz no tiene padres, por lo tanto sus nodos a_I y a_D no existen pero se toman las fracciones $\frac{0}{1}$ y $\frac{1}{0}$ para reemplazarlos. Ojo que $\frac{1}{0}$ no es una fracción, pero al utilizar la caracterización de la desigualdad de fracciones vista antes se tiene que $\frac{1}{0}$ se comporta como una fracción más grande que cualquier otra.

Con esto vemos que $\frac{0}{1} < \frac{1}{1}$ y $0 * 1 < 1 * 1$, o que es similar a que $\frac{1}{1} < \frac{1}{0}$ para efectos de esta domestración. Con esto probando nuestro caso base.

Propuesto, ¿por que es similar? revisar la demostración anterior para ver que $\frac{1}{0}$ no contradice ninguna hipótesis en la parte anterior.

Caso inductivo:

Para esta inducción lo que asumimos es que todos los nodos arriba de a_i son menores que sus nodos a la derecha y mayores a sus nodos de la izquierda.

Hay 2 casos para a_i:

Caso 1: a_i es hijo izquierdo, llamaremos a_p al padre de a_i. Como es hijo izquierdo existe un nodo a_I a la izquierda de a_p tal que:

$\frac{m_i}{n_i} = \frac{m_p+m_I}{n_p+n_I}$

Pero por hipótesis inductiva los nodos a la izquierda de a_p tienen asociados valores menores a los de él. Usando esto y las desigualdades anteriores concluímos:

$\frac{m_i}{n_i} < \frac{m_p}{n_p}$

Y por transitividad es menor que todo lo que está a la derecha de a_p.

Caso 2: a_i es hijo derecho. En este caso vamos a notar que existe un a_D que se utilizó para definir a_i y que es el ancestro más cercano que está a su derecha. Nuevamente a_p es el padre de a_i.

Por hipótesis inductiva la fracción asociada a a_p es menor que la asociada a a_D, por lo tanto:

$\frac{m_i}{n_i} = \frac{m_p+m_D}{n_p+n_D}$

Y podemos concluir que:

$\frac{m_p}{n_p} < \frac{m_i}{n_i} < \frac{m_D}{n_D}$

Con esto vemos que $\frac{m_i}{n_i}$ es menor que lo que está a la derecha de a_D, pero como a_D es el ancestro más cercano hacia la derecha no puede haber ninguna otra fracción a la derecha de a_i y a la izquierda de a_D que no sea un hijo de a_D. Finalmente sabemos que el hijo izquierdo de a_D debe ser ancestro de a_i, porque si no habría otro ancestro más cercano a a_i a la derecha.

De esto concluímos que la única posibilidad es que haya un descendiente por la derecha de a_D con valor asociado menor al de a_i, pero esto no puede ser ya que por hipótesis inductiva los nodos a la derecha de a_D son mayores que él y para construir un hijo derecho se suma con un nodo a la derecha del padre.

Imaginemos que un mechón o mechona quiere imprimir una serie de guías para estudiar, pero tiene tan solo k hojas para imprimir en el CEC. Cada guía g_i tiene una cantidad de hojas h_i que requiere para imprimirse y un valor asociado v_i que representa la cantidad de ayuda al estudio que va a aportar. Considerando que cada guía debe imprimirse completa o no imprimirse, defina un algoritmo recursivo que permita al o la estudiante maximizar el valor total de las guías que imprime.

Solución: Para construir un algoritmo recursivo conviene pensar en los casos fáciles de responder, osea nuestros casos base. Otra cosa que debemos decidir es de que forma vamos a divir el problema para hacerlo manejable.

Claramente un caso base cómodo es cuando tenemos 0 hojas para imprimir en el cec, la respuesta es trivialmente 0.

Otro caso base que nos sirve es cuando tenemos una sola guía, en esta caso la imprimimos si podemos o no hacemos nada si no.

Luego nos falta analizar cómo construir una respuesta a un caso anterior:

Si tenemos la respuesta para n guías y k hojas calcularla para $n+1$ guías y las mismas k hojas equivale a tomar el máximo entre la solución imprimiendo la unueva guía o no imprimiendola:

$S[n+1][k] = \text{máx}(S[n][k], v_{n+1} + S[n][k - h_{n+1}])$

La solución sin imprimirla es $S[n][k]$ porque es lo mismo que el óptimo sin la nueva guía y la solución tomandola es lo mismo que el óptimo de las primeras n guías pero si tenemos menos hojas (ya que gastamos en imprimirlas).

Es claro ver de esto que para resolver el problema vamos a usar las soluciones con s más pequeños.

Aprovechamos que las guías están ordenadas por su índice y podemos generalizar el algoritmo anterior.