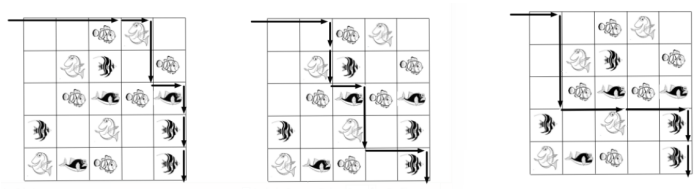
Facultad de Ciencias, UNAM
Análisis de Algoritmos
Tarea 3

Rubí Rojas Tania Michelle

11 de diciembre de 2020

1. Un pescador está sobre un océano rectangular. El valor del pez en el punto (i, j) está dado por un arreglo A de dimensión $2(n \times m)$. Diseña un algoritmo que calcule el máximo valor de pescado que un pescador puede atrapar en un camino desde la esquina superior izquierda a la esquina inferior derecha. El pescador sólo puede moverse hacia abajo o hacia arriba, como se ilustra en la siguiente figura:



   Solución:

   Esto se traduce al siguiente algoritmo:

   -

2. Dados dos árboles generadores T y R de una gráfica G. Muestra cómo encontrar la secuencia más corta de árboles generadores $T_0, T_1, \ldots, T_k$ tal que $T_0 = T, T_k = R$, y cada árbol T_i difiere del anterior T_{i-1} agregando y borrando una arista.

3. Sea G una gráfica con n vértices. Un subconjunto S de los vértices de G es independiente si cualesquiera dos elementos de S no son adyacentes. En general, el problema de encontrar el conjunto independiente de una gráfica es un problema $NP-$completo. Pero en algunos casos, este problema puede resolverse eficientemente. Sea T un árbol con raíz con n vértices. Cada nodo $v \in T$ tiene asociado un peso $w(v)$. Utilizándo programación dinámica, encuentre un algoritmo de tiempo lineal para encontrar el conjunto independiente de T de peso máximo.

4. Mientras caminas por la playa encuentras un cofre de tesoros. El cofre contiene n tesoros con pesos $w_i, \ldots, w_n$ y valores $v_1, \ldots, v_n$. Desafortunadamente, sólo tienes una maleta que sólo tiene capacidad de carga M. Afortunadamente, los tesoros se pueden romper si es necesario. Por ejemplo, la tercera parte de un tesoro i tiene peso $\frac{w_i}{3}$ y valor $\frac{v_i}{3}$.

   - Describe un algoritmo voraz de tiempo $\theta(n \log n)$ que resuelve este problema.
     Solución: Como podemos romper los tesoros, entonces una buena idea para atacar este problema es calcular el valor del costo unitario de cada uno de los n tesoros que encontremos, esto con el objetivo de poder seleccionar aquellos tesoros que nos aportan mayor valor y tienen un menor peso. Luego, ordenamos los tesoros, en orden descendente, de acuerdo a su valor de costo unitario. Lo ordenamos de esta forma para ir tomándo siempre a los tesoros que nos aportan mayor valor en tiempo constante. Después, iremos metiéndo a la mochila lo máximo posible del tesoro con mayor valor unitario, es decir, del tesoro que está al inicio de esta nueva lista ordenada de tesoros. Si se termina este tesoro, es

decir, logramos meterlo completo a la mochila, y queda más lugar en la mochila, entonces tomamos el segundo objeto con mayor valor unitario y repetimos el proceso (vamos metiéndo lo máximo posible del tesoro). Terminamos cuando llenamos por completo la mochila.

Así, esto se traduce al siguiente algoritmo:

1. Obtenemos el valor del costo unitario de cada uno de los n tesoros, es decir, calculamos

$$u_i = \frac{v_i}{w_i} \qquad \text{con } i \in \{1, 2, \ldots, n\}$$

2. Ordenamos los n tesoros, en orden descendente, de acuerdo a su valor de costo unitario u_i. Así, supongamos que esta nueva lista es de la forma

$$l = \{t_1, t_2, \ldots, t_n\}$$

3. Sean M la capacidad de la mochila y c la capacidad actual de la misma en un momento dado. Escogemos el tesoro t_i con $i \in \{1, 2, \ldots, n\}$ de la lista l.
   - Si $c \geq w_i$, entonces metemos al tesoro t_i a la mochila (pues podemos cargar con la totalidad del tesoro) y la capacidad actual es actualizada como $c = c - w_i$.
   - Si $c < w_i$, entonces tomamos únicamente una parte del tesoro t_i. Esta fracción será lo que nos falte para terminar de llenar la mochila, es decir, tomamos la parte $\frac{c_i}{w_i}$ del tesoro. Terminamos.

Este algoritmo funciona porque

Ahora bien, calcular el valor de costo unitario para cada uno de los n tesoros nos toma tiempo lineal, ya que recorremos toda la lista de los n tesoros. Ordenar la lista usando `HeapSort` nos toma $\Theta(n \log n)$. Luego, en el peor de los casos, se verifica que cada uno de los n tesoros entre en la mochila, por lo que esto nos tomará tiempo lineal, pues recorremos toda la lista de los tesoros. Por lo tanto, la complejidad total del algoritmo es $\Theta(n \log n)$.

- ¿Se puede mejorar el tiempo de ejecución de tu algoritmo a $\theta(n)$? Si es un no, explica por qué; si es un sí, menciona el cambio.

  Solución: Sí es posible mejorar la complejidad, y para lograrlo seguiremos el siguiente algoritmo

  1. Obtenemos el valor del costo unitario de los n tesoros, esta vez sin ordenarlos. Así, supongamos que el conjunto de valores unitarios es

$$\rho = \{\frac{v_1}{w_i}, \ldots, \frac{v_n}{w_n}\} = \{u_1, \ldots, u_n\}$$

  2. Encontramos la mediana del conjunto ρ usando el algoritmo Select, visto en clase. Recordemos que Select funciona de la siguiente manera:
     a) Separar a los n elementos en $\lfloor \frac{n}{5} \rfloor$ grupos de 5 elementos cada uno (a lo más un grupo de tamaño $n \pmod 5$).
     b) Encontrar la mediana de cada uno de los $\lfloor \frac{n}{5} \rfloor$ grupos (los cuales están ordenados por inserción y tomamos al elemento de enmedio; si el grupo tiene un número par de elementos, tomamos a la mayor de las medianas)
     c) Usamos Select recursivamente para encontrar la mediana x del conjunto de $\lceil \frac{n}{5} \rceil$ medianas encontradas en el paso anterior (si el conjunto de medianas es de longitud par, entonces tomamos la más pequeña).
     d) Luego, dividimos el conjunto de entrada con elemento pivote x (la mediana de las medianas) usando el algoritmo Partition (del algoritmo QuickSort). Sea k el número de elementos en la parte inferior de la partición, de manera que x es el $k-$ésimo elemento y $n-k$ es el número de elementos en la parte superior.
     e) Si $i = k$, entonces regresamos a x. En otro caso, usamos Select recursivamente para encontrar el $i-$ésimo elemento más pequeño en el lado inferior si $i < k$, o el $(i-k)-$ésimo elemento más pequeño en el lado superior si $i > k$.

Definimos a m como la mediana del conjunto ρ.

3. Creamos tres nuevos conjuntos C_1, C_2, C_3, donde
   - C_1 tendrá los valores unitarios cuyos valores sean estrictamente mayores a la mediana m, es decir,
     $$C_1 = \{u_i \mid m < u_i, 1 \leq i \leq n\}$$
     Este conjunto se refiere a los tesoros que nos conviene tener, pues tienen mayor valor por unidad.
   - C_2 tendrá los valores unitarios cuyos valores sean igual a la mediana m, es decir,
     $$C_2 = \{u_i \mid m = u_i, 1 \leq i \leq n\}$$
   - C_3 tendrá los valores unitarios cuyos valores sean estrictamente menores a la mediana m, es decir,
     $$C_3 = \{u_i \mid m > u_i, 1 \leq i \leq n\}$$

   De esta forma, consideremos dos casos posibles:

   a) Si el valor de la suma de los valores unitarios de los tesoros en el conjunto C_1
   $$\sum_{i \in C_1} w_i$$
   excede la capacidad de la mochila, entonces repetimos recursivamente el algoritmo, pero esta vez sólo lo aplicamos sobre C_1 para quedarnos con los tesoros que valen más.

   b) En otro caso, vamos metiéndo los tesoros que corresponden al conjunto C_2 mientras haya tesoros y no excedamos la capacidad de la mochila.
   1) Si se llena la mochila, entonces vamos a considerar los tesoros correspondientes al conjunto C_1 y a los tesoros que agregamos del conjunto C_2.
   2) En otro caso, actualizamos la capacidad actual de la mochila en
   $$\sum_{i \in C_1} w_i + \sum_{i \in C_2} w_i$$
   pues estamos considerando los tesoros correspondientes a C_1 y C_2. Como todavía hay espacio, entonces volvemos a repetir el algoritmo sobre el conjunto C_3.

Podemos resolver esta recurrencia utilizándo el Teorema Maestro:
La recurrencia

$$T(n) = aT\left(\frac{n}{b}\right) + cn^k$$
$$T(1) = c$$

donde a, b, c, k son constantes, se resuelve como

$$T(n) \in \Theta(n^k) \qquad sia < b^k$$
$$T(n) \in \Theta(n^k \log(n)) \qquad sia = b^k$$
$$T(n) \in \Theta(n^{\log_b(a)}) \qquad sia > b^k$$

Como en nuestro caso tenemos

$$T(n)$$

entonces con $a = 1, b = 2, c = 1, k = 1$ la solución a la recurrencia es $\Theta(n)$.
Por lo tanto, la complejidad total de nuestro algoritmo es $\Theta(n)$.

5. Un grupo de n personas quiere comprar un ramo de m flores, cada flor va a tener un costo asociado c_i, pero si un cliente ha comprado c veces entonces el dueño del puesto le vende la flor i en costo $(c+1)v_i$. Diseña un algoritmo que en tiempo $O(m \log m)$ minimice el costo de comprar todas las flores.

   SOLUCIÓN: Sabemos que entre más flores compre una persona más grande será el costo que tendrá que pagar, por lo que el objetivo es minimizar el número máximo de flores que cualquier persona compra, esto para que las compras se distribuyan lo más uniformemente posible. Además, debemos optimizar el órden en que compramos las flores, por lo que tenemos que comprar primero las flores más caras (el costo adicional que necesitamos pagar después es lineal en c_i). Así, podemos intuir que primero debemos ordenar las flores (en órden descendente) de acuerdo a su costo c_i; y después distribuirlas uniformemente entre las personas que compran las más caras primero. Siguiéndo esta idea, planteamos el siguiente algoritmo:

   1. Ordenamos las m flores, en órden descendente, de acuerdo a su costo asociado c_i. Supongamos que la lista ordenada es de la siguiente forma:

      $$C = \{c_1, c_2, \ldots, c_m\}$$

   2. Definimos una variable `cc=0`, la cual se encargará de llevar el valor del costo de comprar todas las flores. Además, definimos una variable `ad = 0`, la cual se encargará de indicar cuál es el costo adicional que debemos agregarle al costo de nuestras flores actuales.
      - Si el número de flores es menor o igual al número de personas, entonces regresamos la suma del costo de todas las flores, es decir,

        $$cc = c_1 + c_2 + \ldots + c_m$$

      - En otro caso, mientras $i = 0$ sea menor que m hacemos:
        - Sumamos el valor del costo de las primeras m flores dentro de la lista C (pues el valor adicional es 0); luego sumamos el valor del costo de las siguientes m flores, pero multiplicándo el costo de cada una de las flores por $[ad + 1]$ con $i \in \{1, 2, \ldots\}$, y así sucesivamente. Esto se puede modelar con lo siguiente:

          ```
          cc += (ad + 1) * C[i]
          ```

        - Si $i + 1$ (mód n) es igual a cero, esto implica que hemos agregado ya las n flores y debemos aumentar en una unidad nuestro costo adicional.
        - Incrementamos en una unidad a nuestra variable i.

   - Regresamos `cc`.

   Este algoritmo funciona porque siempre garantizamos que las flores con un costo asociado más grande sean las que tengan un valor adicional menor. Como tenemos una lista C con los costos ordenados, entonces podemos acceder a las flores con mayor valor en tiempo constante. El caso trivial es cuando el número de flores m es menor o igual que el número de personas, ya que podemos darle a lo más a cada persona una flor, así que nuestro costo adicional siempre es 0 y esto resulta en sumar los costos de todas las flores m. Luego, cuando $m > n$ lo que hacemos es darle las primeras n flores a cada una de las n personas, donde el costo adicional es de 0 (pues es la primer flor). Posteriormente, le damos las siguientes n flores a cada una de las n personas, lo que implica que nuestro costo adicional aumentará en una unidad y serán más caras que su precio original (pero como está ordenada la lista C entonces sabemos que la suma de estos nuevos costos no es más grande que si hubiéramos multiplicado por este costo adicional actual a la suma de las m flores anteriores). Repetimos este proceso hasta que las flores se hayan terminado, sumándo siempre bloques de n flores con un costo adicional dado y en caso de que lleguemos al punto donde el número de flores que nos quedan sin repartir es menor que el número de personas, entonces simplemente las distribuimos entre las personas que podamos, pero sin perder de vista el valor adicional que les corresponde en ese momento. Así, como las estamos distribuyéndo uniformemente entre las personas, siguiéndo esta idea podemos garantizar que obtenemos el mínimo costo de comprar todas las flores.

   Ahora bien, sabemos que ordenar las flores nos toma $\Theta(m \log m)$. Obtener la suma de los costos de las m flores nos toma $\Theta(m)$, pues siempre debemos recorrer todo el arreglo C. Por lo tanto, la complejidad total del algoritmo es de $\Theta(m \log m)$.

6. Sean $k, n \in \mathbb{N}$. El problema de los huevos, es el siguiente: tenemos un edificio con n pisos y k huevos. Sabemos que hay un piso f tal que si dejamos caer un huevo desde el piso f, se estrellará. Si dejamos caer un huevo desde un piso r tal que $r < f$, el huevo no se estrellará, y si dejamos caer el huevo desde un piso $r \geq f$, el huevo se estrellará (es posible que $f = 1$, en cuyo caso, el huevo siempre se estrellará. Si $f = n+1$, el huevo nunca se estrellará). **Una vez que un huevo se estrella, no lo podemos usar nuevamente**. Si disponemos de k huevos, ¿cuál es el menor número de experimentos (dejar caer un huevo) que se tienen que hacer para determinar a f? Sea $E(k, n)$ el mínimo número de experimentos que tiene que hacer para determinar a f.

   a) Pruebe que $E(1, n) = n$.

   b) Encuentre una recurrencia para $E(k, n)$. Utilice programación dinámica para encontrar $E(k, n)$. ¿Qué tan rápido es su algoritmo?

7. Construye el árbol de Huffman para codificar el siguiente texto:

   "La rabia es como el picante. Una pizca te despierta, pero en exceso te adormece"

   Solución: Primero, ignorándo mayúsculas, vamos a crear una tabla de frecuencias para los símbolos y letras en nuestro texto

símbolo	frecuencia
b	1
u	1
x	1
z	1
.	1
,	1
d	2
l	2
m	2
n	3
s	3
i	4
p	4
r	4
t	4
c	5
o	5
a	8
e	13
␣	14

Figura 1: Tabla de frecuencias ordenada

Luego, realizaremos las actualizaciones de la tabla de frecuencias:

símbolo	frecuencia
x	1
z	1
.	1
,	1
bu	2
d	2
l	2
m	2
n	3
s	3
i	4
p	4
r	4
t	4
c	5
o	5
a	8
e	13
␣	14

Tabla 1: Unimos los símbolos `b` y `u`

símbolo	frecuencia
.	1
,	1
bu	2
xz	2
d	2
l	2
m	2
n	3
s	3
i	4
p	4
r	4
t	4
c	5
o	5
a	8
e	13
␣	14

Tabla 2: Unimos los símbolos `x` y `z`

símbolo	frecuencia
bu	2
xz	2
.,	2
d	2
l	2
m	2
n	3
s	3
i	4
p	4
r	4
t	4
c	5
o	5
a	8
e	13
␣	14

Tabla 3: Unimos los símbolos `.` y `,`

símbolo	frecuencia
.,	2
d	2
l	2
m	2
n	3
s	3
buxz	4
i	4
p	4
r	4
t	4
c	5
o	5
a	8
e	13
␣	14

Tabla 4: Unimos los símbolos `bu` y `xz`

símbolo	frecuencia
l	2
m	2
n	3
s	3
buxz	4
.,d	4
i	4
p	4
r	4
t	4
c	5
o	5
a	8
e	13
␣	14

Tabla 5: Unimos los símbolos `.,` y `d`

símbolo	frecuencia
n	3
s	3
buxz	4
.,d	4
lm	4
i	4
p	4
r	4
t	4
c	5
o	5
a	8
e	13
␣	14

Tabla 6: Unimos los símbolos `l` y `m`

símbolo	frecuencia
buxz	4
.,d	4
lm	4
i	4
p	4
r	4
t	4
c	5
o	5
ns	6
a	8
e	13
␣	14

Tabla 7: Unimos los símbolos `n` y `s`

símbolo	frecuencia
lm	4
i	4
p	4
r	4
t	4
c	5
o	5
ns	6
buxz.,d	8
a	8
e	13
␣	14

Tabla 8: Unimos los símbolos `buxz` y `.,d`

símbolo	frecuencia
p	4
r	4
t	4
c	5
o	5
ns	6
buxz.,d	8
lmi	8
a	8
e	13
␣	14

Tabla 9: Unimos los símbolos `lm` y `i`

símbolo	frecuencia
t	4
c	5
o	5
ns	6
buxz.,d	8
lmi	8
pr	8
a	8
e	13
␣	14

Tabla 10: Unimos los símbolos `p` y `r`

símbolo	frecuencia
o	5
ns	6
buxz.,d	8
lmi	8
pr	8
a	8
tc	9
e	13
␣	14

Tabla 11: Unimos los símbolos `t` y `c`

símbolo	frecuencia
buxz.,d	8
lmi	8
pr	8
a	8
tc	9
ons	11
e	13
␣	14

Tabla 12: Unimos los símbolos `o` y `ns`

símbolo	frecuencia
pr	8
a	8
tc	9
ons	11
e	13
␣	14
buxz.,dlmi	16

Tabla 13: Unimos los símbolos `buxz.,d` y `lmi`

símbolo	frecuencia
tc	9
ons	11
e	13
␣	14
buxz.,dlmi	16
pra	16

Tabla 14: Unimos los símbolos `pr` y `a`

símbolo	frecuencia
e	13
␣	14
buxz.,dlmi	16
pra	16
tcons	20

Tabla 15: Unimos los símbolos `tc` y `ons`

símbolo	frecuencia
buxz.,dlmi	16
pra	16
tcons	20
e ␣	27

Tabla 16: Unimos los símbolos `e` y ␣

símbolo	frecuencia
tcons	20
e ␣	27
buxz.,dlmipra	32

Tabla 17: Unimos los símbolos `buxz.,dlmi` y `pra`

símbolo	frecuencia
buxz·,dlmipra	32
tconse ␣	47

Tabla 18: Unimos los símbolos `tcons` y `e` ␣

buxz·,dlmipransotce ␣	79

Tabla 19: Unimos los símbolos `buxz.,dlmipra` y `tconse` ␣

Así, el árbol de Huffman se vería de la forma:

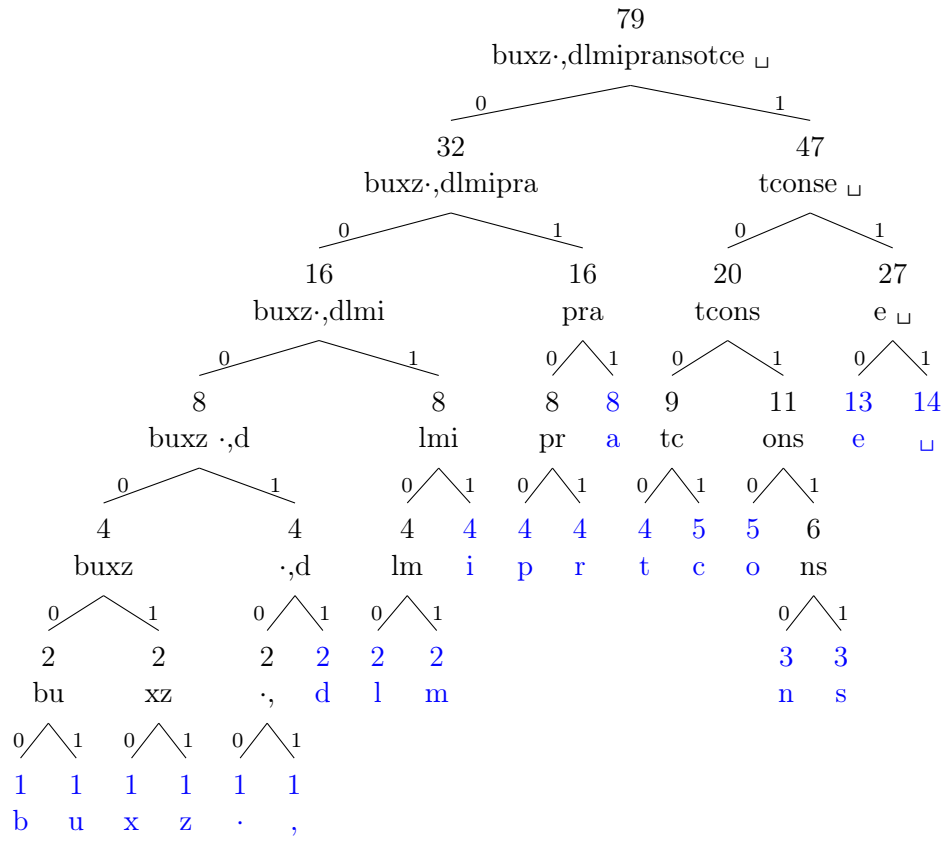


Figura 2: Árbol de Huffman

Por lo tanto, la codificación de cada símbolo sería

símbolo	codificación
b	000000
u	000001
x	000010
z	000011
·	000100
,	000101
d	00011
l	00100
m	00101
i	0011
p	0100
r	0101
a	011
t	1000
c	1001
o	1010
n	10110
s	10111
e	110
␣	111

8. Supongamos que el mago Merlín tiene un conjunto $A[1, \ldots, n]$ de pociones, las cuales puede mezclar de

dos maneras consecutivas con un costo de $A[i] \times A[i+1]$ y resulta en la poción $A[i] + A[i+1]$. Merlín quiere mezclar todas las pociones pero con el mínimo costo.

a) Diseña un algoritmo que garantice unir todas las pociones con un costo mínimo.

b) ¿Cuál es el mínimo costo si se tienen 5 pociones cuyos valores son: $1, 9, 6, 23$?