Ejercicios en clase: Heapsort, Análisis probabilístico, Prog. Dinámica

Análisis y Diseño de Algoritmos

23 de mayo de 2020

Heap

Ejercicio 1. Muestre que un heap con n elementos tiene altura $\lfloor \lg n \rfloor$.

Solución. Por induccion en n. Si $n = 1$ entonces todos los nodos tienen altura $h = 0 = \lfloor \lg 1 \rfloor$.

Suponga ahora que $n > 0$. Note que un heap con n nodos tiene $\lceil n/2 \rceil$ hojas (ejercicio).

Sea un heap con $\lfloor n/2 \rfloor$ nodos resultante de borrar las hojas del heap original. Es claro que este heap tiene menos nodos que el heap original. Luego, por hipótesis de induccción, la altura de este nuevo heap es $\lfloor \lg \lfloor n/2 \rfloor \rfloor$.

Note que la altura del heap original, es igual a la altura de este heap más uno. Es decir $\lfloor \lg \lfloor n/2 \rfloor \rfloor + 1$. Mostraremos que este número es igual que $\lfloor \lg n \rfloor$.

Es claro que

$$\lfloor \lg n \rfloor \leq \lg n < \lfloor \lg n \rfloor + 1.$$

Portanto,

$$2^{\lfloor \lg n \rfloor} \leq n < 2^{\lfloor \lg n \rfloor + 1}.$$

Lo que implica que

$$2^{\lfloor \lg n \rfloor - 1} \leq n/2 < 2^{\lfloor \lg n \rfloor}.$$

Tomando piso a los tres términos, como el primer y tercer términos son números enteros,

$$2^{\lfloor \lg n \rfloor - 1} \leq \lfloor n/2 \rfloor < 2^{\lfloor \lg n \rfloor}.$$

Tomando logaritmo en base dos a los tres términos,

$$\lfloor \lg n \rfloor - 1 \leq \lg \lfloor n/2 \rfloor < \lfloor \lg n \rfloor.$$

Tomando piso a los tres términos, como el primer y tercer términos son números enteros,

$$\lfloor \lg n \rfloor - 1 \leq \lfloor \lg \lfloor n/2 \rfloor \rfloor < \lfloor \lg n \rfloor.$$

Concluimos que

$$\lfloor \lg n \rfloor \leq \lfloor \lg \lfloor n/2 \rfloor \rfloor + 1 < \lfloor \lg n \rfloor + 1.$$

Portanto $\lfloor \lg \lfloor n/2 \rfloor \rfloor + 1 = \lfloor \lg n \rfloor$, como queríamos mostrar.

Ejercicio 2. Muestre que existen como máximo $\lceil n/2^{h+1} \rceil$ nodos de altura h en un heap con n nodos.

Solución. Por induccion en n. Si $n = 1$ entonces todos los nodos tienen altura 0. Luego, si $h = 0$ entonces el heap tiene como máximo $1 \leq \lceil 1/2^{0+1} \rceil$ nodos de altura 0. Y si $h > 0$, la cantidad de nodos a altura h es igual a $0 \leq \lceil n/2^{h+1} \rceil$.

Suponga ahora que $n > 0$. Note que un heap con n nodos tiene $\lceil n/2 \rceil$ hojas (ejercicio). Sea un heap con $\lfloor n/2 \rfloor$ nodos resultante de borrar las hojas del heap original. Es claro que este heap tiene menos nodos que el heap original. Luego, por hipótesis de induccción, el número de nodos a altura $h - 1$ en el nuevo heap es menor o igual a

$$\lceil \lfloor n/2 \rfloor / 2^{(h-1)+1} \rceil \leq \lceil (n/2)/2^h \rceil = \lceil n/2^{h+1} \rceil.$$

Note que un nodo a altura h en el heap original, tiene altura $h - 1$ en el nuevo heap. Portanto, el número de nodos a altura h en el heap original está acotado superiormente por $\lceil n/2^{h+1} \rceil$.

Ejercicio 3. Ilustre la operación MAX-HEAPIFY desde el nodo 2 y desde el nodo 3 en el arreglo $[27, 3, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0]$.

Ejercicio 4. Ilustre la operación BUILD-MAX-HEAP en el arreglo $[5, 3, 17, 10, 84, 19, 6, 22, 9]$.

Ejercicio 5. Ilustre la operación de HEAPSORT en el arreglo $[5, 13, 2, 25, 7, 17, 20, 8, 4]$.

Ejercicio 6. Illustre la operación MAX-HEAP-INSERT$(A, 10)$ en el heap $[15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2]$.

Ejercicio 7. La operación HEAP-DELETE elimina el item en el nodo i del heap A. De una implementación de HEAP-DELETE que corre en $O(\lg n)$ para un max-heap con n elementos.

Solución.

HEAP-DELETE(A, i)
1: **if** $A[i] > A[A.heapsize]$
2: $\quad A[i] = A[A.heapsize]$
3: $\quad$ MAX-HEAPIFY (A, i)
4: **else**
5: $\quad$ HEAP-INCREASE-KEY$(A, i, A[A.heapsize])$
6: $A.heapsize = A.heapsize - 1$

Ejercicio 8. Escriba en seudocódigo cada uno de los siguientes procedimientos, que implementan una fila de prioridades con un min-heap: HEAP-MINIMO, HEAP-EXTRAE-MINIMO, HEAP-DISMINUYE-LLAVE y HEAP-INSERTA.

Análisis probabilístico / Quicksort

Ejercicio 9. Sea X una variable aleatória que guarda el número de caras en dos lanzamientos de uma moneda justa. ¿Cuánto vale $E[X^2]$? ¿Cuánto vale $E[X]^2$?

$E[X] = \sum_{x=0}^{2} x \cdot Pr(X = x) = 2 \cdot 1/4 + 1 \cdot 1/2 + 0 \cdot 1/4 = 1$. Luego $E^2[X] = 1$.
$E[X^2] = \sum_{x=0}^{2} x^2 \cdot Pr(X = x) = 2^2 \cdot 1/4 + 1^2 \cdot 1/2 + 0 \cdot 1/4 = 3/2$.

Ejercicio 10. En el pseudocódigo de HIRE-ASSISTANT, suponiendo que los candidatos se presentan en de manera aleatoria uniforme, ¿cual es la probabilidad que se contrate exactamente una vez?, ¿cual es la probabilidad de que se contrate n veces?

Recordemos

```
HIRE-ASSISTANT(n)
1  best = 0          // candidate 0 is a least-qualified dummy candidate
2  for i = 1 to n
3      interview candidate i
4      if candidate i is better than candidate best
5          best = i
6          hire candidate i
```

Figura 1: Tomada del libro Cormen, Introduction to Algorithms

Para que se contrate una única vez, el mejor candidato se presenta primero. En otros términos, es la probabilidad de que en un arreglo de n elementos, el mayor elemento aparezca al inicio, esto es,

$$(n-1)!/n! = 1/n.$$

Si se contrata n veces, el candidato actual debe ser el mejor entre los que han sido entrevistados. En otras palabras, en un arreglo de n elementos, el máximo del subarreglo $A[1..i]$ está en la i-ésima posición. Esto solo ocurre si el arreglo está ordenado de manera creciente. Luego, la probabilidad pedida es

$$1/n!$$

Ejercicio 11. Considere el siguiente algoritmo que determina el el mayor y menor elemento de un vector $v[1 \ldots n]$ con números positivos distintos.

```
MAYORMENOR(v, n)
1: mayor = v[1]
2: menor = v[1]
3: for i = 2 to n
4:     if v[i] > mayor
5:         mayor = v[i]
6:     else
7:         if v[i] < menor
```

8: $menor = v[i]$
9: **return** $mayor, menor$

Suponga que la entrada del algoritmo es una permutación de 1 a n escogida uniformemente dentre todas las permutaciones de 1 a n. ¿Cual es el número esperado de comparaciones ejecutadas en la línea 7 del algoritmo? ¿Cual es el número esperado de atribuiciones efectuadas en la línea 8 del algoritmo?

Sea X la variable aleatoria que cuenta el número de ejecuciones de la línea 7 del algoritmo.
Para cada $i = 2, \ldots, n$, sea X_i la variable aleatoria que mide el número de ejecuciones de la línea 7 en la correspondiente iteración. Note que

$$\begin{aligned} E[X_i] &= Pr(X_i = 1) \\ &= Pr(\text{“se ejecuta la linea 7 en la iteracion correspondiente a } i\text{”}) \\ &= Pr(\text{“}A[i] \text{ no es máximo en } A[1..i]\text{”}) \\ &= 1 - Pr(\text{“}A[i] \text{ es máximo en } A[1..i]\text{”}) \\ &= 1 - 1/i \end{aligned}$$

Luego

$$E[X] = \sum_{i=2}^{n} E[X_i] = \sum_{i=2}^{n} (1 - 1/i) = \sum_{i=1}^{n} (1 - 1/i) = n - H_n \approx n - \log n.$$

Sea Y la variable aleatoria que cuenta el número de ejecuciones de la línea 8 del algoritmo.
Para cada $i = 2, \ldots, n$, sea Y_i la variable aleatoria que mide el número de ejecuciones de la línea 8 en la correspondiente iteración. Note que

$$\begin{aligned} E[Y_i] &= Pr(Y_i = 1) \\ &= Pr(\text{“se ejecuta la linea 8 en la iteracion correspondiente a } i\text{”}) \\ &= Pr(\text{“}A[i] \text{ es mínimo en } A[1..i]\text{”}) \\ &= 1/i \end{aligned}$$

Luego

$$E[Y] = \sum_{i=2}^{n} E[Y_i] = \sum_{i=2}^{n} (1/i) = \sum_{i=1}^{n} (1/i) - 1 = H_n - 1 \approx \log n - 1.$$

Ejercicio 12. Escriba una función que reciba un vector con n letras A y B y, a través de intercambios, mueve todas las A al comienzo del vector. Su función deberá tener tiempo de ejecución $O(n)$.

Ejercicio 13. ¿Cual es el tiempo de ejecución de QUICKSORT cuando todos los elementos del arreglo A tienen el mismo valor?

Programación Dinámica

Ejercicio 14. Diseñe un algoritmo que tome como parámetros un número entero $n > 0$ y un vector que almacene un secuencia de n enteros y devuelve la longitud de una subsecuencia creciente de suma máxima. Su función debe consumir tiempo $O(n^2)$. Modifique su función (sin empeorar su complejidad asintótica) para que también devuelva una subsecuencia creciente de suma máxima.

Sea $A[1..n]$ el arreglo en cuestión. Para cada i, sea $OPT_\ell(i)$ la longitud de una subsecuencia de suma máxima que termina en i. Para cada i, sea $OPT_s(i)$ la suma de elementos de una subsecuencia de suma máxima que termina en i. Para cada i, sea

$$M(i) = \max_{j=1}^{i-1}\{OPT_s(j) : A[j] \leq A[i]\}$$

Tenemos la siguientes recurrencias para OPT_ℓ, OPT_s:

$$OPT_s(i) = \begin{cases} A[1] & \text{si } i = 1 \\ A[i] + M(i) & \text{si } i > 1 \end{cases}$$

$$OPT_\ell(i) = \begin{cases} 1 & \text{si } i = 1 \\ 1 + OPT_\ell(k), \text{ donde } k < i, A[k] \leq A[i] \text{ y } OPT_s(k) = M(i) & \text{si } i > 1 \end{cases}$$

Podemos diseñar el siguiente algoritmo de programación dinámica.
Recibe: Un arreglo $A[1..n]$ de números enteros
Devuelve: La longitud de una subsecuencia creciente de suma máxima en A

MAX-TAM-SCSM(A, n)
1: $s[1] = A[1]$
2: $\ell[1] = 1$
3: **for** $i = 2$ **to** n
4: $\quad s[i] = A[i]$
5: $\quad l[i] = 1$
6: $\quad$ **for** $j = 1$ **to** $i - 1$
7: $\quad\quad$ **if** $A[j] \leq A[i]$ AND $s[j] + A[i] > s[i]$
8: $\quad\quad\quad s[i] = s[j] + A[i]$
9: $\quad\quad\quad k = j$
10: $\quad \ell[i] = \ell[k] + 1$
11: Sea i^* tal que $s[i^*] = \max\{s[i] : 1 \leq i \leq n\}$
12: **return** $\ell[i^*]$

Ejercicio 15. Debes cortar un tronco en varias piezas. La empresa la mejor forma de hacerlo es con Analog Cutting Machinery (ACM), que cobra según el longitud de registro a cortar. Su máquina de corte permite que solo se haga un corte a la vez. Si queremos hacer varios cortes, es fácil ver que diferentes órdenes de estos cortes conducen a precios muchos diferentes. Por ejemplo, considere un registro de 10 metros de largo, que debe ser cortado a

2, 4 y 7 metros de uno de sus extremos. Hay varias posibilidades. Podemos primero cortar a 2 metros, luego a 4 y después a 7. Este pedido cuesta $10 + 8 + 6 = 24$, porque el primer tronco tenía 10 metros de largo, lo que quedaba era de 8 metros de largo y el la última pieza era de longitud 6. Si cortamos en orden 4, luego 2, luego 7, pagaríamos $10 + 4 + 6 = 20$, que es más barato. Su jefe ordenó un programa que, dada la longitud l del tronco y k puntos $p_1, \ldots, p_k$ de corte, encuentre el costo mínimo para realizar estos cortes en el ACM.

Sea ℓ la longitud total del tronco y sea $p[1..n]$ el arreglo que guarda los n puntos de corte (en el ejemplo $\ell = 10$ y $p = [2, 4, 7]$). Sea $OPT(i, j)$ el valor de un corte óptimo del subtronco que comienza en p_i y termina en p_j y considera todos los puntos entre p_i y p_j. Queremos $OPT(0, n + 1)$, donde $p_0 = 0$ y $p_{n+1} = \ell$. Considere la siguiente recurrencia para $OPT(i, j)$.

$$OPT(i,j) = \begin{cases} 0 & \text{si } j = i+1 \\ \min_{k=i+1}^{j-1}\{OPT(i,k) + OPT(k,j)\} + p_j - p_i & \text{si } i < j+1 \end{cases}$$

Podemos diseñar el siguiente algoritmo de programación dinámica.

Recibe: Un arreglo $p[1..n]$ de n números enteros que representan puntos de corte y una longitud $\ell > k$

Devuelve: El costo minimo para realizar los cortes

MIN-COST-CORTES(p, n, ℓ)

```
1: for i = 0 to n
2:     M[i][i + 1] = 0
3: for i = n − 1 to 0
4:     for j = i + 2 to n + 1
5:         M[i][j] = ∞
6:         for k = i + 1 to j − 1
7:             if M[i][k] + M[k][j] + p[j] − p[i] < M[i][j]
8:                 M[i][j] = M[i][k] + M[k][j] + p[j] − p[i]
9: return  M[0][n]
```

Ejercicio 16. Los transbordadores se utilizan para transportar automóviles entre una orilla de un río y la otra. Considere los transbordadores miden lo suficiente como para acomodar dos carriles de automóviles a lo largo de toda su longitud. Los autos entran a los carriles por un lado del transbordador y salen, en la otra orilla, al otro lado del transbordador.

La cola de automóviles para ingresar al transbordador es una sola línea y el operador dirige cada automóvil a cualquiera de las dos pistas del transbordador, el carril izquierdo o el carril derecho, a manera de equilibrar las dos pistas. Cada automóvil en la cola tiene una longitud diferente, que el operador estima mientras los autos están en línea. Con base en estas estimaciones, el operador decide cuál de los dos carriles cada automóvil debe abordar y abordar tantos automóviles en la cola como sea posible. Escribir un programa que informa al operador a qué carril debe dirigir cada automóvil para maximizar el número de carros cargados en el transbordador.

Sea $A[1..n]$ el arreglo con la longitud de los n automoviles, sea ℓ la longitud del transbordador. Considere que la lista de espera de los automóviles es $A[1], A[2], \ldots, A[n]$. Sea

$OPT(\ell_1, \ell_2, i)$ el maximo de automóviles que pueden entrar en un transbordador con filas con tamaños disponibles ℓ_1 y ℓ_2, considerando los automóviles $A[i], A[i+1], \ldots, A[n]$. El problema pide $OPT(\ell, \ell, 1)$. Considere la siguiente recurrencia:

$$OPT(\ell_1, \ell_2, i) = \begin{cases} 0 & \text{si } A[i] > \ell_1, A[i] > \ell_2 \\ 1 + OPT(\ell_1, \ell_2 - A[i], i-1) & \text{si } A[i] > \ell_1, A[i] \leq \ell_2 \\ 1 + OPT(\ell_1 - A[i], \ell_2, i-1) & \text{si } A[i] \leq \ell_1, A[i] > \ell_2 \\ 1 + \text{máx}\{OPT(\ell_1 - A[i], \ell_2, i-1), OPT(\ell_1, \ell_2 - A[i], i-1)\} & \text{caso contrário} \end{cases}$$

Podemos diseñar el siguiente algoritmo de programación dinámica.

Recibe: Un arreglo $A[1..n]$ de n números enteros que representan las longitudes de los autos y una longitud ℓ del transbordador.

Devuelve: La mayor cantidad de vehículos que entran en el transbordador

MAX-CANT-AUTOS(A, n, ℓ)

```
1: for ℓ1 = 0 to ℓ
2:     for ℓ2 = 0 to ℓ
3:         for i = 0 to n
4:             if A[i] > ℓ1 AND A[i] > ℓ2
5:                 M[ℓ1][ℓ2][i] = 0
6: ℓ[1] = 1
7: for ℓ1 = 0 to ℓ
8:     for ℓ2 = 0 to ℓ
9:         for i = 0 to n
10:            if A[i] > ℓ1 AND A[i] ≤ ℓ2
11:                M[ℓ1][ℓ2][i] = M[ℓ1 − A[i]][ℓ2][i − 1] + 1
12:            if A[i] ≤ ℓ1 AND A[i] > ℓ2
13:                M[ℓ1][ℓ2][i] = M[ℓ1][ℓ2 − A[i]][i − 1] + 1
14:            if A[i] ≤ ℓ1 AND A[i] ≤ ℓ2
15:                M[ℓ1][ℓ2][i] = máx{M[ℓ1 − A[i]][ℓ2][i − 1], M[ℓ1][ℓ2 − A[i]][i − 1]} + 1
16: return M[0][n]
```

Obs, el algoritmo anterior puede ser mejorado, ya que $\ell_1 + \ell_2 + \sum_{j=1}^{i-1} A[j]$, por lo tanto puede ser hecho con una matriz (dos dimensiones).

Ejercicio 17. Supongamos que tienes un máquina y un conjunto de n trabajos, identificados por los números $1, 2, ..., n$, para procesar en esa máquina. Cada trabajo j tiene un tiempo de procesamiento t_j, un beneficio p_j y una fecha límite fin d_j. La máquina solo puede procesar un trabajo a la vez, y el trabajo debe realizarse ininterrumpidamente durante t_j unidades de tiempo consecutivas. Si el trabajo ya está completado en su fecha límite d_j, usted recibe una ganancia p_j, pero si se completa después de su fecha límite, no recibe ganancias. Escriba un algoritmo para encontrar el orden de ejecución de los trabajos que maximiza la suma de beneficios, suponiendo que todos los tiempos de procesamiento son enteros entre 1 y n. ¿Cuál es el tiempo de ejecución de su algoritmo?

Supongamos sin perdida de generalidad que $d_1 < d_2 < \ldots < d_n$. Mostraremos que siempre existe una solución óptima para el problema que ejecuta el primer trabajo o bien al inicio o bien al final. Sea $S = (i_1, i_2 \ldots, i_n)$ una solución optima cualquiera (una permutación de $1, 2, \ldots, n$).

Suponga que $i_k = 1$ para $1 < k < n$. Si $\sum_{j=1}^{k} t_{i_j} \leq d_1$, es decir el trabajo 1 acaba a tiempo, entonces cualquier trabajo que acaba antes del trabajo 1 también acaba a tiempo. Portanto, podemos permutar el orden de estas tareas, encontrando una solución óptima que comienza en el trabajo 1. Si $\sum_{j=1}^{k} t_{i_j} > d_1$, es decir el trabajo 1 no acaba a tiempo, entonces la solución resultante de enviar el trabajo 1 al final y permutar el trabajo 1 y los demás trabajos que le sigan, tiene gananica mayor o igual que la solución original, portanto también es óptima.

Luego, basta concentrarnos en este tipo de soluciones. Sea $OPT(r, s, i)$ que considera la asignación de los trabajos $\{i..n\}$ en el intervalo $[r..s]$, donde el trabajo i aparece o al inicio o al final. Note que buscamos $OPT(0, \sum_{j=1}^{n} t_j, 1)$. Tenemos la siguiente recurrencia para OPT:

$$OPT(r, s, i) = \begin{cases} 0 & \text{si } r = s \\ \text{máx}\{p_i + OPT(r + t_i, s, i + 1), OPT(r, s - t_i, i + 1)\} & \text{si } r + t_i \leq d_i \\ \text{máx}\{OPT(r + t_i, s, i + 1), OPT(r, s - t_i, i + 1)\} & \text{si } r + t_i > d_i \end{cases}$$

Hacer el algoritmo a partir de la recurrencia.

Ejercicio 18. En China, la gente usa pares de palillos para comer, pero el Sr. L es un poco diferente. El utiliza tres palillos de dientes: un par y uno extra largo para recoge objetos más grandes pegándolos. La longitud de los dos palillos de dientes normales más pequeños debe ser lo más cerca posible, pero la longitud del palillo extra no importa mientras sea el más largo de los tres. Para un conjunto de palillos de dientes de longitud a, b y c ($a \leq b \leq c$), la función $(a - b)^2$ mide qué tan malo es el conjunto. El Sr. L invitó a k personas a su fiesta de cumpleaños y está ansioso por presentar su forma de usar palillos de dientes. Debe preparar k juegos de palillos de dientes. Pero los palillos de dientes del Sr. L son longitudes variadas!

Escribe una función que, dado k y las longitudes ordenadas de manera no decreciente de cada uno de los $n \geq 3k$ palillos, encuentre una manera de componer los k conjuntos de palillos de dientes para minimizar la suma de lo malos que son los conjuntos.