Universidad de Chile
Departamento de Ciencias de la Computación
Control 2 CC40A
Semestre: Agosto 2009
Solution Usage FileNames

Problema 1 (1.5 puntos)
File: `./seleccion01.pbtex`
Usage:

- *2009 Septiembre, Universidad de Chile, DCC, Control 2. Average: X/X*

1. Muestre que el segundo elemento en rango (e.g. b) de un conjunto ordenado (e.g. $\{a, b, c, d, e, f\}$) dado e n un areglo desordenado (e.g. $[d, c, a, b, e]$) se puede obtener en $n + \lceil \log_2 n \rceil - 1$ comparaciones. *Hint:* Usa un árbol de tornamiento (y no un heap).

——————————begin solution——————————

Obviamente, no es acceptable de buscar dos veces para el min, que costaria $2n - 2$ comparaciones. Se puede hacer mucho mejor.

- Solución en $n + \lceil \log_2 n \rceil - 1$ comparaciones:
  a) Composa un arbol de tornamiento con n horas.
  b) Calcula el min en cada nodo interno, marcando el hijo "ganador" en cada nodo interno.
  c) Eso costa $n - 1$ comparaciones, y da el min m del arreglo.
  d) Calcula el min de los que perdieron contra m, siguiendo la raiz hasta la hoja que contiene m: eso costa al maximo $\lceil \lg n \rceil$ comparaciones.
  e) En total costa $n - 1 + \lceil \lg n \rceil$.
- Solución en $2(n - \lfloor \lg n \rfloor) - 1$ comparaciones (que no es acceptable):
  a) Calcula el heap del arreglo en $2(n - \lceil \lg n \rceil - 1)$ comparaciones. Es bien conocido que la complejidad de heapify es $O(n)$, pero menos conocido porque. Una manera eleganta de calcular la cantidad de comparaciones exacta de heapify es de mostrar una bijection entre los caminos de los elementos en el heap durante la "heapification", y todos los caminos que van arriba de un node, una vez a la izquierda y el resto del tiempo a la derecha. La cantidad de comparaciones del algoritmo es dos veces la cantidad de arestas de estos caminos. Como los caminos estan de interseccion vacilla, es facil de contarlas las arestas: coloran todo el arbol falta el camino que va de la raiz a la hoja mas a la izquierda, que contiene $\lfloor \lg n \rfloor$. La cantidad total de aristas es $n - 1$ aristas en el arbol, menos $\lfloor \lg n \rfloor$ aristas en el camino derecho, sumando a $n - \lfloor \lg n \rfloor - 1$ aristas y $2(n - \lfloor \lg n \rfloor - 1)$ comparaciones.
  b) Returna el minimo de los dos hijos de la raiz.
  c) El costo total es $2(n - \lfloor \lg n \rfloor) - 1$ comparaciones.

——————————end solution——————————

2. Extienda el algoritmo para hallar una fórmula para el k-ésimo. ¿Cuándo usaría este algoritmo?

——————————begin solution——————————

- Solucion con familia de arboles en $O(n + k \lg n)$:
  a) Construya el árbol T_1 que represente el torneo hecho entre los n elementos. m_1 es el elemento ganador encontrado en $n-1$ comparaciones.
  b) Construya el árbol T_2 que represente el torneo hecho entre los $\lg |T_1| = \lg n$ elementos que "perdieron"contra m_1. m_2 es el elemento ganador, encontrado en $\lceil \lg n \rceil - 1$ comparaciones, que es menos que $\lg n$.
  c) Por induccion, supponga que
    - los arboles $T_1, \ldots, T_{i-1}$ estan calculados;
    - $\forall j$ el arbol T_j tiene altura menos que $2 \lg n$;
    - el $i-1$-ésimo elemento m_{i-1} es calculado.

    Construya el árbol T_i que represente el torneo hecho entre los elementos que "perdieron"contra m_{k-1} en cada de los arboles $T_1, \ldots, T_{i-1}$ (siguiendo en cada arbol la rama iniciando en la hoja m_{i-1} hasta el nodo donde m_{i-1} perdi).
    Este arbol contiene al maximo $2i \lg n$ elementos, porque los arboles estan de altura al maxima $2 \lg n$ por induccion: su altura es menos que $\lceil \lg(2i \lg n) \rceil$ que es menos que $1 + \lceil \lg i \rceil + \lceil \lg \lg n \rceil$, que es menos que $2 \lg n$
    m_i es el elemento ganador, encontrado en $2i \lg n$ comparaciones (addicionales) al maximo.
  d) Asi el costo total para calcular m_k es $n - 1 + \lg n + 2 \sum_{i=3}^{k} i \lg n$, que es menos que $n - 1 + (4k + 1) \lg n \in O(n + k \lg n)$.
- Fusioning the tree
  - Build the tournament tree T with n leaves.
  - Compute the minimum m_1 in $n-1$ comparisons, noting the result of each comparisons in the nodes of the tree.
  - For i going from 2 to k,
    - Divide T in $\lg n$ trees, which roots correspond to the elements which lost against m_{i-1}. The heights of those trees are roughly all distinct power of twos (minus one).
    - Merge those trees into a single tournament tree by building a tournament tree on their roots, weighted by the size of their subtree. The height of this tree is no more than $\lg n$.
    - Compute the minimum m_i in $\lg n$ comparisons, noting the result of each comparisons in the nodes of the tree.
  - Each additional step costs $\lg n$ comparisons, which yield the total cost of $n - 1 + k \lg n \in O(n + k \lg n)$

Esto algoritmo es interessante solamente cuando k es muy pequeño en comparacion con n, de manera que el costo total es menos que el costo del algoritmo lineal k-select.

——————————end solution——————————

3. En clase se mostró el algoritmo para obtener la mediana de n elementos en tiempo $O(n)$ mediante particiones de 5 elementos. Vea qué ocurre si en vez de 5 se usan 3 elementos. ¿Y si

se usan 7? Halle una fórmula de costo para $2k+1$ elementos, con $k > 1$, suponiendo que la mediana para los $2k+1$ elementos cuesta $d(2k+1)$.

————————begin solution————————

- $$T(n) = T(\frac{n}{2k+1}) + T(n\frac{3k+1}{2(2k+1)}) + \frac{n}{2k+1}d(2k+1)$$

- Por H.I. $T(n) \leq c.n$
  - $$c(\frac{n}{2k+1}) + c(n\frac{3k+1}{2(2k+1)}) + \frac{n}{2k+1}d(2k+1) \leq c.n$$
  - $$\frac{2d(2k+1)}{k-1} \leq c$$

- Para particiones de 3 elementos ($k = 1$) el comportamiento ya no es lineal. Para particiones de 7 elementos ($k = 3$) la constante se hace menor, pero el aporte de d(2k+1) se incrementa.

————————end solution————————

Problema 2 (1.5 puntos)
File: `./biasedCoin.tex`
Usage:

- *2009 Septiembre, Universidad de Chile, DCC, Control 2. Average: X/X*
- *Assignment 3, Year 2006, Spring Term, School of Computer Science at the University of Waterloo.*
- *Spring 2006, University of Waterloo, SCS, Assignment 3 CS240:*

  maxpoints *10.00*
  max *10.00*
  median *5.00*
  average *5.15*

1. Implementamos "skip list" con probabilidad $1/t$ de crecer una torre. Si $t = 3$, ¿cual es la altura en promedio $E(h)$ de un "skip-list" tal que cada torre tiene altura de al menos uno?

   Hint: El valor de $S = \sum_{h=1}^{\infty} \frac{h}{3^h}$ se puede calcular estudiando la serie geométrica que aparece en la diferencia $3S - S$.

   ——————————begin solution——————————

   The first step in this problem is to determine the probability that a tower will reach height h. By definition, $P(h \geq h_0) = (1/3)^{h_0-1}(2/3)$. The *expectation* of h is found by considering each value of h and its associated probability of occurrence. Specifically,

$$\begin{aligned} E(h) &= 1 \cdot P(h = 1) + 2 \cdot P(h = 2) + 3 \cdot P(h = 3) + \ldots \\ &\leq 1 \cdot P(h \geq 1) + 2 \cdot P(h \geq 2) + 3 \cdot P(h \geq 3) + \ldots \\ &= \sum_{h=1}^{\infty} h \left(\frac{1}{3}\right)^{h-1} \left(\frac{2}{3}\right) \\ &= 2 \sum_{h=1}^{\infty} \frac{h}{3^h} \end{aligned}$$

   where in the last line we factored out a 2 and combined fractions.

   For the sake of simplicity, let

$$S = \sum_{h=1}^{\infty} \frac{h}{3^h} \tag{1}$$

   where we will "remember" that our desired answer is $2S$. If we can determine the expression for S, we are essentially done. As suggested in the hint, the mathematical "trick" to finding an expression for S is to consider the difference $3S - S$. Let us write out the first few terms of $3S$ and S to determine the pattern.

$$\begin{aligned} 3S &= 1 + \frac{2}{3} + \frac{3}{3^2} + \frac{4}{3^3} + \ldots \\ S &= \quad \frac{1}{3} + \frac{2}{3^2} + \frac{3}{3^3} + \ldots \end{aligned}$$

When we subtract S from $3S$ with respect to common denominators, we have

$$\begin{aligned} 3S - S &= 1 + \left(\frac{1}{3} + \frac{1}{3^2} + \frac{1}{3^3} + \ldots\right) \\ &= 1 + \sum_{h=1}^{\infty} \frac{1}{3^h} \end{aligned}$$

where the last summation is a geometric series with $a = \frac{1}{3}$ and $r = \frac{1}{3}$. Since $\frac{1}{3} < 1$, the geometric series is convergent, and

$$\begin{aligned} \sum_{h=1}^{\infty} \frac{1}{3^h} &= \frac{\frac{1}{3}}{1 - \frac{1}{3}} \\ &= \frac{1}{2} \end{aligned}$$

and so we can conclude that

$$\begin{aligned} 3S - S &= 1 + \frac{1}{2} \\ 2S &= \frac{3}{2} \end{aligned}$$

where we recall that the desired solution is in terms of $2S$. Therefore, $E(h) = \frac{3}{2}$.

————————end solution————————

2. El costo en promedio de una búsqueda es mejor para $t = 3$ que para $t = 2$, pero es peor para $t = 5$ en comparación con $t = 2$. (De hecho, es óptimo para $t = e$, la constante de Euler.) Considerando que la altura en promedio de la torre decrece cuando t crece, explique por qué el costo de la búsqueda no decrece cuando t crece. (No esperamos análisis matemático, pero una corta explicación cualitativa.)

   Hint: Su explicación no debería ser mas larga que página media.

————————begin solution————————

The expected cost of a search is the expected number of downward steps (bounded by the maximum height of all the towers) PLUS the expected number of forward steps. It is true that the expected number of downward steps goes down as t increases, but the expected number of forward steps does not.

To put it another way, if too many towers are tall, then there will many long searches in high level lists, but if the towers are too small, then there will many long searches in low level lists: it is only logical that the cost of search is not monotone.

————————end solution————————

Problema 3 (1.5 puntos)
File: `./insertValues02.tex`

Usage:

- *2009 Septiembre, Universidad de Chile, DCC, Control 2. Average: X/X*
- *University of Waterloo: CS240, Assignment 4w, Fall 2002*
- *University of Waterloo: CS240, Assignment 4, Winter 2006; average mark: 8.95/10*

Suponga que las paginas de memoria secundaria tienen capacidad de 3 llaves. Muestre el estado de un sistema de "extendible hashing" después de la inserción de cada de las siguientes llaves.

$$00001, 10000, 10001, 01001, 01111, 00000, 10010, 11111, 10111.$$

————————begin solution————————

$\emptyset \rightarrow$ | ^{0}00001, 10000, 10001 |

$0 \rightarrow$ | ^{1}00001, 01001, 01111 |
$1 \rightarrow$ | ^{1}10000, 10001 |

$00 \rightarrow$ | ^{2}00001, 00000 |
$01 \rightarrow$ | ^{2}01001, 01111 |
$10 \rightarrow$ | ^{2}10000, 10001, 10010 |
$11 \rightarrow$ | ^{2}11111 |

$000 \rightarrow$ | ^{2}00001, 00000 |
$001 \rightarrow$ |
$010 \rightarrow$ | ^{2}01001, 01111 |
$011 \rightarrow$ |
$100 \rightarrow$ | ^{3}10000, 10001, 10010 |
$101 \rightarrow$ | ^{3}10111 |
$110 \rightarrow$ | ^{2}11111 |
$111 \rightarrow$ |

————————end solution————————

Problema 4 (1.5 puntos)
File: `./preguntas03.extex`
Usage:

- *2009 Septiembre, Universidad de Chile, DCC, Control 2. Average: X/X*

1. Define los siguientes terminos (incluya formulas)

   - "Competitive Ratio"

     ————————begin solution————————

     - Optimizacion/approximacion
       - A es $k(n)$ competitive para un problema de *minimizacion* si

         $$\exists b, \forall n, x|x| = n,\ C_A(x) - k(n)C_{OPT}(x) \leq b$$

       - A es $k(n)$ competitive para un problema de *maximisacion* si

         $$\exists b, \forall n, x|x| = n,\ C_{OPT}(x) - k(n)C_A(x) \leq b$$

     - Competitive Ratio
       - an algoritmo en linea es c-competitive si

         $$\exists \alpha, \forall I - ALG(I) \leq cOPT(I) + \alpha$$

       - an algoritmo en linea es strictamente c-competitive si

         $$\forall I - ALG(I) \leq cOPT(I)$$

     ————————end solution————————

   - Familia "2-universal".

     ————————begin solution————————

     Una familia H de funciones de hashing de $[0, M-1]$ a $[0, N-1]$ se dice "2-universal" si

     $$\forall x \neq y \in [0, M-1],\ \Pr_{h \in H}(h(x) = h(y)) \leq 1/N$$

     Donde h es definido por

     $$\forall x, yC_{x,y} = 1sih(x) = h(y), y0sino$$

     La idea es que elegir h al azar a dentro de H va a dar una buena funcion de hash con alta probabilidad.

     ————————end solution————————

   - $p(n)$-approximación

     ————————begin solution————————

- Dado un problema de minimización, un algoritmo A es un $p(n)$-approximacion si $\forall n \, \text{máx} \frac{C_A(x)}{C_{OPT}(x)} \leq p(n)$
- Dado un problema de maximización, un algoritmo A es un $p(n)$-approximacion si $\forall n \, \text{máx} \frac{C_{OPT}(x)}{C_A(x)} \leq p(n)$

————————end solution————————

2. Considere un algoritmo Monte Carlo para un problema, cuyo costo de ejecución es $T(n)$ en el peor caso, y que produce una respuesta correcta con probabilidad $p(n)$. Suponga que dada una solución, el costo de verificar si es correcta o no es $C(n)$. Muestre que existe un algoritmo Las Vegas (o sea, nunca se equivoca) cuyo costo promedio es $O((T(n) + C(n))/p(n))$.

————————begin solution————————

- Aplicar el algoritmo Monte Carlo y verificar si la solución es correcta: $O(T(n) + C(n))$.
- Se repite el procedimiento hasta encontrar una solución correcta.
- El número total de iteraciones es una variable aleatoria con distribución binomial negativa, por lo tanto el tiempo esperado es $\frac{tiempo\_iteracion}{probabilidad\_exito} = O((T(n) + C(n))/p(n))$

————————end solution————————

3. Diseñe un algoritmo lineal para el problema del vertex-cover cuando el grafo es en realidad un árbol (grafo acíclico).

————————begin solution————————

- Haga un recorrido del árbol en profundidad para hallar el vertex cover V'.
- Al terminar de explorar el subárbol de un nodo, si éste no pertenece a V', significa que es un nodo hoja o que todos sus hijos ya pertenecen a V', entonces se marca a su padre como perteneciente.
- Si un nodo ya pertenece a V', no se hace nada.

————————end solution————————