Illinois Institute of Technology
Department of Computer Science

Solutions to Second Examination

CS 430 Introduction to Algorithms
Spring, 2018

Wednesday, March 7, 2018
10am–11:15am & 11:25am–12:40pm
111 Robert A. Pritzker Science Center

Exam Statistics

XXX students took the exam; there were XX no-shows recorded as 0. The range of scores was XX–XX, with a mean of XX.XX (this excludes the no-shows), a median of XX, and a standard deviation of XX.XX. Very roughly speaking, if I had to assign final grades on the basis of this exam only, above XX would be an A (XX), XX–XX a B (XX), XX–XX a C (XX), XX–XX a D (XX), below XX an E (XX). Every student should have been able to get almost full credit on the first and fourth problems, plus a dozen or so points on the remaining problems; no score should have been below 50.

Problem Solutions

1. We saw in class in Lecture 7 (Janauary 31), that the external path length can equivalently be defined recursively as

$$\begin{aligned} \mathrm{EPL}(\square) &= 0 \\ \mathrm{EPL}(x) &= \mathrm{EPL}(x.\textsc{left}) + \mathrm{EPL}(x.\textsc{right}) + n + 1 \end{aligned}$$

   where $x.\textsc{left}$ and $x.\textsc{right}$ are the left and right subtrees, respectively, of x, and n is the number of internal nodes in the subtree rooted at x. Thus $\mathrm{EPL}(x)$ does not satisfy the hypothesis of Theorem 14.1 (page 346 in CLRS), namely that $\mathrm{EPL}(x)$ depend only on $\mathrm{EPL}(x.\textsc{left})$ and $\mathrm{EPL}(x.\textsc{right})$; hence we cannot conclude that it can be maintained in a red-black tree. But Theorem 14.1 only gives us *sufficient conditions* for maintainence, not *necessary conditions.*

   If we knew the number of internal nodes in the subtree rooted at x, then $\mathrm{EPL}(x)$ would satisfy the hypothesis of Theorem 14.1. So, denote by $\textsc{size}(x)$ the number of internal nodes in the subtree rooted at x. $\textsc{size}(x)$ can be written recursively as

$$\begin{aligned} \textsc{size}(\square) &= 0 \\ \textsc{size}(x) &= \textsc{size}(x.\textsc{left}) + \textsc{size}(x.\textsc{right}) + 1, \end{aligned}$$

   and hence $\textsc{size}(x)$ satisfies the hypothesis of Theorem 14.1 and can be maintained in a red-black tree. Thus by maintaining $\textsc{size}(x)$ we can also maintain $\mathrm{EPL}(x)$ in a red-black tree.

2. (a) Let S_{ij} be the largest sum that can be reached from row i, diagonal j. Then,

$$S_{ij} = \begin{cases} T_{ij} & \text{if } i = N \\ T_{ij} + \max\left(S_{i+1,j}, S_{i+1,j+1}\right), & \text{if } i < N \end{cases}$$

The largest sum on a path from the apex to the bottom is S_{00}.

(b) Let t_{ij} be the time to compute S_{ij}. Then,

$$t_{ij} = \begin{cases} O(1) & \text{if } i = N \\ O(1) + t_{i+1,j} + t_{i+1,j+1}, & \text{if } i < N \end{cases}$$

Induction on i proves that $t_{N-i,k} = \Theta(2^i)$ for all i.

(c) Memoizing the computed values of S_{ij} means that $O(1)$ time is required for each t_{ij}, assuming the row below has already been computed. There are about $N^2/2$ elements in the triangular array, so the time with memoization is $O(N^2)$.

(d) To keep track of the path giving the largest sum, we add a direction to the memo: d_{ij} is "left" or "right" according to which element is the max in the second line of the equation for S_{ij}.

3. (a) Here are three jobs for which the stated greedy method schedules 1 job, and clearly that is the best possible because the three jobs all conflict:

i	1	2	3
s_i	3	2	1
f_i	4	4	4

(b) The greedy algorithm always gives the optimum schedule: You can either argue parallel to the proof in the text (or lecture) or just transform the jobs $\{(s_i, f_i)\}$ to the mirror image set of jobs $\{(-f_i, -s_i)\}$. Then by the proof for the finishing-time-first version, the greedy algorithm always gives the optimum schedule.

4. Take the potential function to be $\Phi(D_i) = 2b_i$, *twice* the number of 1-bits after the ith increment. Then, paralleling the computation at the bottom of page 461 in CLRS3,

$$\Phi(D_i) - \Phi(D_{i-1}) \leq 2[(b_{i-1} - t_i) - b_{i+1}] = 2(1 - t_i).$$

and the amortized cost of an increment operation is

$$\begin{aligned} \hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= (t_i + 1 + T(i)) + \Phi(D_i) - \Phi(D_{i-1}) \\ &\leq t_i + 1 + t_i + 2(1 - t_i) \\ &= 3. \end{aligned}$$

Thus the amortized time remains $O(1)$ even with significant time wasting in line 9.