Illinois Institute of Technology
Department of Computer Science

Solutions to Second Examination

CS 430 Introduction to Algorithms
Fall, 2014

Wednesday, October 29, 2014
10am–11:15pm, 121 Life Sciences
11:25am–12:40pm, 113 Stuart Building

Exam Statistics

90 students took the exam. 7 students were absent; their zeros are not counted in the statistics that follow. The range of scores was 3–90, with a mean of 44.01, a median of 42, and a standard deviation of 18.79. Very roughly speaking, if I had to assign final grades on the basis of this exam only, above 60 would be an A (21), 40–59 a B (25), 20–39 a C (39), 10–20 a D (3), below 10 an E (2).

Problem Solutions

1. Yes, nodes can be augmented with $F(x)$ because $F(x)$ satisfies the hypothesis of Theorem 14.1 (page 346 in CLRS3), namely that $F(x)$ depends only on $f(x)$ (stored in the node), $F(\textsc{left}(x))$, and $F(\textsc{right}(x))$:

$$F(x) = \begin{cases} f(x) & \text{if } x \text{ has no children,} \\ f(x) + F(\textsc{left}(x)) & \text{if } x \text{ has only a left child,} \\ f(x) + F(\textsc{right}(x)) & \text{if } x \text{ has only a right child,} \\ f(x) + F(\textsc{left}(x)) + F(\textsc{right}(x)) & \text{otherwise.} \end{cases}$$

2. (a) The Principle of Optimality tells us

$$C[i,j] = \begin{cases} c_{1,1} & \text{if } i = j = 1, \\ c_{1j} + C[1, j-1] & \text{if } i = 1, j > 1, \\ c_{i1} + C[i-1, 1] & \text{if } i > 1, j = 1, \\ c_{ij} + \max(C[i-1, j], C[i, j-1]) & \text{otherwise.} \end{cases}$$

   (b) The array can be filled in row by row (column by column also works) in constant time per entry because to fill in $C[i,j]$ we just need to have filled in the squares above and to the left. Thus the following algorithm takes time $\Theta(n^2)$:

```
1: for j = 1 to n do
2:   for i = 1 to n do
3:     if i = j = 1 then
4:       C[i, j] ← c_ij
5:     else if i = 1, j > 1 then
6:       C[i, j] ← c_1j + C[1, j − 1]
7:     else if i > 1, j = 1 then
8:       C[i, j] ← c_i1 + C[i − 1, 1]
9:     else
10:      // i > 1, j > 1
11:      if C[i − 1, j] > C[i, j − 1] then
12:        C[i, j] ← c_ij + C[i − 1, j]
13:      else
14:        C[i, j] ← c_ij + C[i, j − 1]
15:      end if
16:    end if
17:  end for
18: end for
```

(c) To determine the sequence of steps resulting in the maximum amount of money, we need to keep track of the cell from which we arrived at cell i, j—the one above or the one to the left. Hence we need another array $D[i, j]$ in which each entry can be "↑" for "above", "←" for "left", of "S" for "start":

```
1: for j = 1 to n do
2:   for i = 1 to n do
3:     if i = j = 1 then
4:       C[i, j] ← c_ij
5:       D[i, j] ← "S"
6:     else if i = 1, j > 1 then
7:       C[i, j] ← c_1j + C[1, j − 1]
8:       D[i, j] ← "↑"
9:     else if i > 1, j = 1 then
10:      C[i, j] ← c_i1 + C[i − 1, 1]
11:      D[i, j] ← "←"
12:    else
13:      // i > 1, j > 1
14:      if C[i − 1, j] > C[i, j − 1] then
15:        C[i, j] ← c_ij + C[i − 1, j]
16:        D[i, j] ← "←"
17:      else
18:        C[i, j] ← c_ij + C[i, j − 1]
19:        D[i, j] ← "↑"
20:      end if
21:    end if
```

22: **end for**
23: **end for**

(d) Let the unmemoized algorithm use time $T(i,j)$ to fill in $C[i,j]$. Then mirroring the recursive definition of $C[i,j]$ we have

$$T(i,j) = \begin{cases} 1 & \text{if } i = 1 \text{ or } j = 1, \\ T(i-1,j) + T(i,j-1) & \text{otherwise.} \end{cases}$$

This is really a lower bound because we are ignoring the $O(1)$ work to compute $C[i,j]$ from $C[i-1,j]$ and $C[i,j-1]$. The recurrence for $T(i,j)$ is thus the recurrence for the binomial coefficients (Pascal's Triangle), so that $T(i,j) = \binom{i-1+j-1}{i-1}$. Thus the total time required to fill in $C[n,n]$ is at least the central binomial coefficient $\binom{2n-2}{n-1}$ which grows like $4^n/\sqrt{n}$ as $n \to \infty$ by Stirling's formula—you did not need to give this growth rate to get the extra credit, just the binomial coefficient.

3. (a) Number the students 1, 2, ..., n. A greedy approach is to increase as much as possible the number of exam questions known to some student after each email message is sent. Student 1 sends his question to student 2, who then sends the two questions she knows to student 3, who then sends the 3 questions he knows to student 4, and so on, until student $n-1$ sends her $n-1$ known questions to student n who then knows all n questions; so far $n-1$ email messages have been sent. Now student n sends the full set of n questions to each of the other $n-1$ students, another $n-1$ email messages. The total number of messages is thus $2n-2$.

Another approach would be to designate one student as the *focus*; the other $n-1$ students send their questions to the focus, whereupon the focus sends all n questions out to the other $n-1$ students. There is a total of $2n-2$ messages sent. This is not a greedy algorithm, but it was an acceptable answer.

(b) At least $2n-2$ email messages are necessary by induction: For 1 student, clearly $0 = 2 \times 1 - 2$ messages are needed. For 2 students, at least $2 \times 2 - 2 = 2$ email messages are needed because each must learn the other's question; this makes the general case clear—if $2(n-1)-2$ are necessary for $n-1$ students, an additional student will increase the minimum number of email messages by 2, one email message for the new student to share his question with somebody and another email message for the new student to receive the other $n-1$ questions from somebody else. Thus the algorithms in part (a) are optimal.

4. Consider the sequence of $2^i - 2$ INCREMENT operations so the counter then has the value $2^i - 1$ and the rightmost i bits all ones and the rest zeros. An INCREMENT must set all these i one-bits to zero and the change the $(i+1)$st bit from zero to one, giving the counter the value 2^i. Now a DECREMENT operation must set all these i zero-bits to zero and the change the $(i+1)$st bit from one to zero and the counter is back to value $2^i - 1$.

If we follow the initial sequence of $2^i - 2$ INCREMENT operations by 2^{i-1} INCREMENT-DECREMENT pairs, we have a total of $2^i - 2 + 2 \times 2^{i-1} = 2^{i+1} - 2$ operations, requiring $(i+1)2^{i-1}$ bit changes. Let $n = 2^{i+1} - 2$; then $i + 1 = \lg(n+2)$ and $2^{i-1} = (n+2)/4$, so the n operations require $O(n \log n)$ bit changes, a lower bound on the amortized time.

5. (a) The proof of New Lemma 19.1 is *exactly* the same as the first two sentences of the proof of the original version.

   (b) The proof of New Lemma 19.4, which includes the observation that $s_0 = 1$ and $s_1 = 2$, begins the same as the proof of the original version, up to the phrase "To bound s_k, we count…". Then we continue: To bound s_k, we count one for z itself and add the degrees of its children:

$$\text{SIZE}(x) \geq s_k \geq 1 + \sum_{i=1}^{k} s_{\text{DEGREE}(y_i)} \geq 1 + \sum_{i=1}^{k} s_{i-1} = 1 + \sum_{i=0}^{k-1} s_i.$$

   It now follows by induction that $s_k \geq 2^k$.

   (c) Because $\text{SIZE}(x) \geq 2^{\text{DEGREE}(x)}$, taking logarithms base 2 we get $\lg \text{SIZE}(x) \geq \text{DEGREE}(x)$, so an n-element (modified) Fibonacci heap has $D(n) \leq \lfloor \log n \rfloor$.