Illinois Institute of Technology
Department of Computer Science

Solutions to Second Examination

CS 430 Introduction to Algorithms
Spring, 2017

Wednesday, March 8, 2017
10am–11:15am, 002 Herman Hall
11:25am–12:40pm, 104 Rettaliata Engineering Center

Exam Statistics

125 students took the exam. The range of scores was 5–97, with a mean of 57.22, a median of 58, and a standard deviation of 23.48. Very roughly speaking, if I had to assign final grades on the basis of this exam only, above 80 would be an A (20), 64–80 a B (33), 40–63 a C (51), 21–39 a D (16), below 21 an E (5). Every student should have been able to get at least half credit on the first problem, full credit (or nearly so) on the third and fourth problems, plus a few points on problem two; that is, no score should have been below 60.

Problem Solutions

1. (a) Let $C[i, j]$ be the minimum cost to reach square i, j from square $1, 1$. The Principle of Optimality tells us

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

(b) Let T_{ij} be the time to fill $C[i, j]$. Each of the squares not on the top row or left column takes one addition to fill in, once the squares above and to the left are filled in; that is, $(n-1)^2$ additions. The recurrence for the remaining additions parallels the recursion for $C[i, j]$:

$$T_{ij} = \begin{cases} 1 & \text{if } i = j = 1, \\ T_{i-1,j} + T_{i,j-1} & \text{otherwise.} \end{cases}$$

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.

(c)

```
for $j = 1$ to $n$ do
  for $i = 1$ to $n$ do
    if $i = j = 1$ then
      $C[i, j] \leftarrow c_{ij}$
    else if $i = 1, j > 1$ then
      $C[i, j] \leftarrow c_{1j} + C[1, j - 1]$
    else if $i > 1, j = 1$ then
      $C[i, j] \leftarrow c_{i1} + C[i - 1, 1]$
    else
      // $i > 1, j > 1$
      if $C[i - 1, j] < C[i, j - 1]$ then
        $C[i, j] \leftarrow c_{ij} + C[i - 1, j]$
      else
        $C[i, j] \leftarrow c_{ij} + C[i, j - 1]$
      end if
    end if
  end for
end for
```

(d) The array is filled in column by column 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 algorithm takes time $\Theta(n^2)$.

(e) To determine the sequence of steps resulting in the minimum cost 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":

```
for $j = 1$ to $n$ do
  for $i = 1$ to $n$ do
    if $i = j = 1$ then
      $C[i, j] \leftarrow c_{ij}$
      $D[i, j] \leftarrow$ "S"
    else if $i = 1, j > 1$ then
      $C[i, j] \leftarrow c_{1j} + C[1, j - 1]$
      $D[i, j] \leftarrow$ "↑"
    else if $i > 1, j = 1$ then
      $C[i, j] \leftarrow c_{i1} + C[i - 1, 1]$
      $D[i, j] \leftarrow$ "←"
    else
      // $i > 1, j > 1$
      if $C[i - 1, j] < C[i, j - 1]$ then
        $C[i, j] \leftarrow c_{ij} + C[i - 1, j]$
        $D[i, j] \leftarrow$ "←"
      else
```

```
18:             $C[i,j] \leftarrow c_{ij} + C[i,j-1]$
19:             $D[i,j] \leftarrow$ "$\uparrow$"
20:           end if
21:         end if
22:       end for
23:     end for
```

2. (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.

3. As in CLRS3, we use the potential function

$$\Phi(n) = \text{the number of 1 bits in the binary representation of } n.$$

The analysis of the INCREMENT operation is just as in the text on page 461; it has amortized cost 2. The calculation of the amortized cost for the RESET operation is similar: As on page 461, suppose that the counter has t 1-bits; the RESET operation sets all t bits to zero. The actual cost of the operation is t. The change potential is $0-t$ because after the reset there are no 1-bits. Hence the amortized cost for the RESET operation is 0.

4. (a) We do a MAKE-SET operation for each student, then we do a UNION for each "same dormitory" constraint, then we do FIND-SET operations for each in a "different dormitory" constraint. If any of the FIND-SET operations has the two students in the same set, the constraints cannot be satisfied:

```
1: for $i := 1, 2, \ldots, n$ do
2:   MAKE-SET($i$)
```

```
3: end for
4: for $i := 1, 2, \ldots, m$ do
5:     Union($s_i, t_i$)
6: end for
7: for $i := 1, 2, \ldots, k$ do
8:     if Find-Set($u_i$) = Find-Set($v_i$) then
9:         return Constraints cannot be satisfied
10:    end if
11: end for
12: return Constraints can be satisfied
```

(b) If we use weighted union and path compression, according to Theorem 21.14 on page 581 of CLRS3 each of the operations Make-Set, Union, Find-Set uses $\alpha(n)$ amortized time, so the sequence of $n + m + 2k$ operations use time $O((n + m + k)\alpha(n))$.

Because section 21.4 was only suggested reading, the following analysis also received full credit: There are n Make-Set operations, m Union operations, and $2k$ Find-Set operations. According to Theorem 21.1 on page 566 of CLRS3, with weighted union, each of the Make-Set operations is worst-case time $O(1)$; the Union and Find-Set operations are worst-case time $O(\log n)$, so the sequence of $n + m + 2k$ operations are worst-case time $O(n + (m + k) \log n)$.