CS 1510: Homework 6

Kyra F. Lee — kfl5@pitt.edu
Zach Sadler — zps6@pitt.edu
John Hofrichter — jmh162@pitt.edu

September 11, 2013

Greedy Algorithm

Problem 11

Assume, to yield a contradiction, that there exists an input I for which the greedy algorithm does not yield optimal output. Let $G(I)$ be the greedy algorithm's output, and $O(I)$ be the optimal schedule that agrees with $G(I)$ for the most initial rows.

Let r_k be the first row where $G(I)$ and $O(I)$ differ. That is, there exist at least one pair of columns c_i and c_j such that $G(I)$ has a 1 in (r_k, c_i) but not in (r_k, c_j), and $O(I)$ has a 1 in (r_k, c_j) but not in (r_k, c_i).

Since r_k is the first row where they disagree, we know that both c_i and c_j still need to be assigned at least one more 1 in row r_{k-1}. By the definition of the greedy algorithm, we know that c_j does not need more 1s than c_i after r_{k-1}, and, because c_j and not c_i is assigned a 1 in r_k in $O(I)$, c_i must need strictly more 1s than c_j after r_k in $O(I)$.

Construct a new output, $O'(I)$, by copying $O(I)$, but swapping the 1s in all pairs of (r_k, c_j)s and (r_k, c_i)s so that r_k in $O'(I)$ agrees with r_k in $G(I)$. Because we know that for every c_i, c_j pair, c_i needs more 1s than c_j, there must be at least one row $r_m, m > k$ where (r_k, c_i) but not (r_k, c_j) has a 1 in $O(I)$. For every c_i, c_j pair, swap the 0 and 1 in r_m too.

$O'(I)$ agrees with $G(I)$ for one more row than $O(I)$ does, because there is at least one (r_k, c_j), (r_k, c_i) swap in r_k.

For every set of affected locations, $(r_k, c_i), (r_k, c_j), (r_m, c_i), (r_m, c_j)$, we have swapped 1s from (r_k, c_j) and (r_m, c_i) to (r_k, c_i) and (r_m, c_j). This diagonal switch swap maintains the number of 1s in all affected rows and columns. Thus, we show that if $O(I)$ is optimal, then $O'(I)$ is as well.

Contradiction, for we assumed that $O(I)$ was the optimal solution that agreed with $G(I)$ for the maximum number of initial rows.

Therefore, the greedy algorithm yields an optimal solution for all inputs. QED.

Dynamic Programming

Problem 1a

A naive recursive implementation which does not save values as it solves smaller problems would have to recalculate each value $T(n)$ at each point. So to calculate $T(n+1)$ would require:

$$\begin{aligned}
T(n+1) &= \sum_{i=1}^{n+1-1} T(i)T(i-1) \\
&= \sum_{i=1}^{n} T(i)T(i-1) \\
&= T(n)T(n-1) + \sum_{i=1}^{n-1} T(i)T(i-1) \\
&= T(n)T(n-1) + T(n) \\
&\geq T(n) + T(n) \\
&= 2T(n)
\end{aligned}$$

That is, $T(n+1)$ requires twice as many calculations as $T(n)$, and so the program is exponential in n.

Problem 1b

If we count the number of operations we have for $T(n)$, we can see that we have

$$\begin{aligned}
T(n) &= \sum_{i=1}^{n-1} T(i)T(i-1) \\
&= T(n-1)T(n-2) + T(n-2)T(n-3) + \cdots + T(1)T(0)
\end{aligned}$$

so we have $n-1$ summands, each of which is two terms multiplied together, which is $O(n)$ operations total. However, it requires $O(n)$ operations to generate each of the $T(i)$ terms which we use in the sum (and we generate these terms exactly once each), so we have $O(n^2)$ operations total.

Problem 1c

A simpler algorithm which uses only $O(n)$ arithmetic operations is a dynamic programming solution:

$T(n)$:
$T[0] = T[1] = 2$
for $i = 2 \cdots n-1$ do
    $T[i] = T[i-1] * T[i-2] + T[i-1]$
end for
return $T[n-1]T[n-2] + T[n-1]$