CS 577 Homework 5

Alex Pizzuto

Question One: 6.4

a Our goal is to disprove the correctness of the following algorithm:

```
for i from 1 to n do
    if N_i < S_i then
        Output "NY in Month i";
    else
        Output "SF in Month i";
    end
end
```

We will disprove this via counterexample: Suppose we have the given table of costs, and the moving cost $M = 500$:

	Month 1	Month 2	Month 3	Month 4
NY	1	1	5	5
SF	5	5	1	2

Then this algorithm would have you choose $[NY, NY, SF, SF]$, thus incurring a cost of $1 + 1 + 500 + 1 + 2 = 505$. However, by inspection, we see we incur a lesser cost by just staying in New York: $[NY, NY, NY, NY] \rightarrow 12$.

b Next, we find an example of an instance in which every optimal plan must move at least three times. Suppose $n = 5$ and $M = 5$:

	Month 1	Month 2	Month 3	Month 4	Month 5
NY	1	1000	1	1000	1
SF	1000	1	1000	1	1000

Then in this example, it is clear that the optimal solution is $[NY, SF, NY, SF, NY]$. This is because the cost of moving plus the cost of the city we were not already in in every case is less than the cost of staying in the city. In other words, if we do not move 4 times, we will incur a cost of at least 1000, whereas our optimal solution has a cost of $1 + 5 + 1 + 5 + 1 + 5 + 1 + 5 + 1 = 25$.

c Now, we find an efficient algorithm that takes values for n, M and sequences of operating costs $N_1, \ldots, N_n$ and $S_i, \ldots, S_n$ and returns the *cost* of the optimal plan. Our goal, before writing out the algorithm, is to determine a recursive goal that we can then optimize from. For this, we recognize that we will either finish the n^{th} month in NY or in SF. In that case, we will have paid either NY_n or SF_n and any relevant moving cost, if necessary. Let us use $opt_{NY}(i)$ to denote the optimal plan that resulted in ending in NY after i months, and let us use $opt_{SF}(i)$ to denote the optimal plan that resulted in ending in SF after i months. We can then divide our problem into two branches, and return the maximum of them at the end, in other words, our goal is to find

$$opt(n) = \min\Big(opt_{NY}(n), opt_{SF}(n)\Big)$$

where in each step, we have (just looking at NY):

$$opt_{NY}(i) = N_n + \min(opt_{NY}(n-1), opt_{SF}(n-1) + M)$$

With this formalism, we can write out our algorithm:

$opt_{NY}(0) = 0$;
$opt_{SF}(0) = 0$;
for *i from 1 to n* **do**
  $opt_{NY}(i) = N_i + \min(opt_{NY}(i-1), opt_{SF}(i-1) + M)$;
  $opt_{SF}(i) = S_i + \min(opt_{SF}(i-1), opt_{NY}(i-1) + M)$;
end
Return $\min\Big(opt_{NY}(n), opt_{SF}(n)\Big)$

Algorithm 1: Minimum Cost City Scheduling Algorithm

Each iteration in this algorithm has a constant number of arithmetic operations and comparisons, and thus the entire algorithm is just $\mathcal{O}(n)$.

Question Two

Our goal is to design an $\mathcal{O}(n^3)$ algorithm using dynamic programming to find an optimal assembly order for combining intervals with associated costs. We know that we must pay $f(1, n)$ as the last step no matter what order we choose prior to that. This will help us establish our recursion. Thus, we define $opt(i, j)$ as the optimal (smallest) cost of assembling the interval $[i, \ldots, j]$. Then, $\forall i < j$,

$$opt(i, j) = \min_{i \le l \le j}\Big(opt(i, l) + opt(l + 1, j)\Big) + f(i, j)$$

We can now implement this formalism into an algorithm:

M[0,. . . ,n] - denotes the matrix used for memoization
set all diagonal elements of M to 0;
for *j from 1 to n* **do**
  for *i less than j* **do**
    $opt(i, j) = \min_{i \le l \le j}\Big(opt(i, l) + opt(l + 1, j)\Big) + f(i, j)$;
    Set M[i,j] = $opt(i, j)$;
  end
end

Algorithm 2: Segment Concatenation Algorithm

We will prove that this is the correct algorithm by doing induction on the size of the interval, $j - i = k$. As our basis of induction, in the case of $i = j$, $(k = 0)$, we know that the cost is 0 as there is no cost to do no joins, and as we cannot have any negative costs, we know this is optimal. For our inductive step, assume that $\exists n > 0$ such that the optimal recurrence equation given is accurate. Now, consider the case of $k = n + 1$. Then, there must be some integer, m, such that the last step in assembling this set joins the m^{th} and $m + 1$ segments. Then the segments $i, \ldots, m$ and $m + 1, \ldots, j$ both have size less than k, and thus were created optimally by our inductive hypothesis. Our algorithm would select the optimal m, and thus our inductive hypothesis assures us that our relation is valid for $k = n + 1$.

Finally, we make note of the running time. There are $\mathcal{O}(n^2)$ loop iterations, each of which is an $\mathcal{O}(n)$ operation. Thus, our overall complexity is $\mathcal{O}(n^3)$.

Question Three: 6.12

Our goal is to find a polynomial-time algorithm to find a configuration of minimum total cost for placing copies of files on different servers where our cost is a combination of the placement costs and the access costs.

We will use $opt(j)$ to denote the optimal (minimum cost) solution $0, \ldots, j$, conditional on the fact that we have selected to put a copy of the file at j. Then, we get

$$opt(j) = c_j + \min_{0 \leq i \leq j} \left(opt(i) + \sum_{i}^{j} access_i \right)$$

We note that the sum of the access costs is just $\binom{j-1}{2}$. It is clear that our goal is to iterate up to $opt(n)$, as we know we must have a copy of the file at n, thus, we can formalize all of this in an algorithm:

M[0,. . . ,n] - array for memoization
M[0] = 0 ;
for *i from 1 to n* **do**
  $opt(i) = c_i + \min_{0 \leq l \leq i} \left(opt(l) + \binom{i-l}{2} \right)$;
  Set M[i] = $opt(i)$
end
return M[n]

Algorithm 3: Copy File Server Algorithm

This algorithm iterates over the servers once, and each iteration is linear in time, thus our overall complexity is $\mathcal{O}(n^2)$, which is polynomial as desired.