Homework 6

Aleck Zhao

October 19, 2017

1 File Replication (33 points)

Suppose we want to replicate a file over n servers $S_1, S_2, \ldots, S_n$. Placing a copy of the file at server S_i results in a *placement cost* of c_i, where c_i is an integer larger than 0.

If a user requests the file from server S_i, and no copy of the file is present at S_i, then the servers $S_{i+1}, S_{i+2}, S_{i+3}, \ldots$ are searched in order until a cope of the file is finally found, say at server S_j, where $j > i$. This results in an *access cost* of $j - i$. (Note that the lower-indexed servers $S_{i-1}, S_{i-2}, \ldots$ are no consulted in this search). The access cost is 0 if S_i holds a copy of the file. We will require that a copy of the file be placed at server S_n, so that all such searches will terminate, at the latest, at S_n.

We'd like to place copies of the files at the servers so as to minimize the sum of placement and access costs. We say that a *configuration* is a choice, for each server S_i with $i = 1, 2, \ldots, n-1$, of whether to place a copy of the file at S_i or not (recall that a copy is always placed at S_n). Equivalently, a configuration is a subset of $\{S_1, S_2, \ldots, S_{n-1}\}$ which consists of the servers at which we place a copy of the file (other than S_n). The *total cost* of a configuration is the sum of all placement costs for servers with a copy of the file, plus the sum of all access costs associated with all n servers.

Give an $O(n^2)$-time algorithm to find a configuration of minimum total cost. As always, prove correctness and running time.

Proof. Let $X_i = \{S_i, S_j, \cdots, S_n\} \subset \{S_i, S_{i+1}, \cdots, S_n\}$ be the optimal configuration on $\{S_i, S_{i+1}, \cdots, S_n\}$ that includes S_i, where $j > i$ the next index is minimized. Let $c(S)$ be the cost associated with a subset $S \subset \{S_0, S_1, S_2, \cdots, S_n\}$. We have $c(X_n) = c_n$, as expected. For convention, set $c_0 = 0$, and we wish to find X_0 minimizing $c(X_0)$. We claim that $X_i \setminus \{S_i\} = X_j$.

Suppose otherwise. Since $X_i \setminus \{S_i\} \subset \{S_j, S_{j+1}, \cdots, S_n\}$, we must have $c(X_j) < c(X_i \setminus \{S_i\})$ since X_j is the optimal configuration containing S_j on $\{S_j, S_{j+1}, \cdots, S_n\}$. Let $\hat{X}_i = \{S_i\} \cup X_j$. Since j is the next server after i in $\hat{X}_i$ and in X_i, there is an access cost for each of S_{i+1} to S_{j-1}, the sum being $1 + 2 + \cdots + (j - i - 1)$. Thus,

$$\begin{aligned} c(\hat{X}_i) &= c_i + [1 + 2 + \cdots + (j - i - 1)] + c(X_j) \\ &< c_i + [1 + 2 + \cdots + (j - i - 1)] + c(X_i \setminus \{S_i\}) \\ &= c(X_i) \end{aligned}$$

a contradiction since X_i was supposed to be optimal on $\{S_i, \cdots, S_n\}$. Now, since j is the next index that minimizes $c(X_i)$, we have

$$\begin{aligned} c(X_i) &= c_i + \min_{i<j\leq n} \{[1 + 2 + \cdots + (j - i - 1)] + c(X_j)\} \\ &= c_i + \min_{i<j\leq n} \left\{ \frac{(j-i)(j-i-1)}{2} + c(X_j) \right\} \end{aligned} \tag{1}$$

Thus, to find the minimum cost, we have the algorithm

```
Config(costs, n) {
    M[n] = costs[n]

    for (i from n-1 to 0) {
        M[i] = costs[i] + min((j-i)*(j-i-1)/2 + M[j])
    }
}
```

The total number of computations the min function computes over all loops is

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2} = O(n^2)$$

so the running time is $O(n^2)$, as desired. To prove correctness of $M[0]$, proceed by induction. Let $OPT(i)$ be the cost of the optimal configuration of $\{S_i, S_{i+1}, \cdots, S_n\}$. The base case is obvious, since $M[n] = c_n = OPT(n)$. Now suppose $M[k] = OPT(k)$ for some $0 < k \leq n$. Then we have

$$\begin{aligned} M[k-1] &= c_{k-1} + \min_{k-1<j\leq n} \left\{ \frac{(j-k)(j-k+1)}{2} + M[j] \right\} \\ &= c_{k-1} + \min_{k-1<j\leq n} \left\{ \frac{(j-k)(j-k+1)}{2} + OPT(j) \right\} \\ &= OPT(k-1) \end{aligned}$$

from (1), as desired. Finally, $M[0]$ is the cost of the optimal configuration, since including S_0 costs 0. □

2 Mobile Business (34 points)

Let's say that you have a great idea for a new food truck, and in order to save money you decide to run it out of your RV so you can live where you work. Each day i there is some demand for your food in Baltimore and some demand in Washington – let's say you would make B_i dollars by being in Baltimore and W_i dollars by being in Washington. However, if you wake up in one city (due to being there the previous day) and want to serve in the other city, it costs you M dollars to drive there.

The goal in this problem is to devise a maximum-profit schedule. A schedule is simply an assignment of locations to days – for each day i, the schedule says whether to serve in Baltimore or Washington. The profit of a schedule is the total profit you make, minus M times the number of times you have to move between cities. For the starting case, you can assume that on day 1 you wake up in Baltimore.

For example, let $M = 10$ and suppose that $B_1 = 1, B_2 = 3, B_3 = 20, B_4 = 30$ and $W_1 = 50, W_2 = 20, W_3 = 2, W_4 = 4$. Then the profit of the schedule ⟨Washington, Washington, Baltimore, Baltimore⟩ would be $W_1+W_2+B_3+B_4-2M = 100$, where one of the M's comes from driving from Baltimore to Washington on day 1, and the other comes from driving from Washington to Baltimore and day 3. The profit of the schedule ⟨Washington, Baltimore, Baltimore, Washington⟩ would be $W_1+B_2+B_3+W_4-3M = 50+3+20+4-30 = 47$.

Given the fixed driving cost M and profits $B_1, \ldots B_n$ and $W_1, \ldots, W_n$, devise an algorithm that runs in $O(n)$ time and computes the profit of an optimal schedule. As always, prove correctness and running time.

Proof. Let $S_i = s_1 s_2 \cdots s_i$ be the optimal schedule for $\{B_1, B_2, \cdots, B_i\}$ and $\{W_1, W_2, \cdots, W_i\}$, where $s_i = B$ and $s_j \in \{B, W\}$ for $1 \leq j < i$. Similarly, let $T_i = t_1 t_2 \cdots t_i$ be the optimal schedule, but where $t_i = W$. Let $p(S)$ be the profit of a schedule S. We claim that for $1 < i \leq n$,

$$p(S_i) = B_i + \max\{p(S_{i-1}), p(T_{i-1}) - M\}$$
$$p(T_i) = W_i + \max\{p(S_{i-1}) - M, p(T_{i-1})\}$$

where $p(S_1) = B_1$ and $p(T_1) = W_1 - M$. If $p(T_i)$ and $p(S_i)$ are the respective optimal schedules, then $p(S_{i+1})$ ends in B, so either s_i is B or it is W. If it is B, then the profit is $p(S_i) + B_{i+1}$ and likewise if it is W, then the profit is $p(T_i) + B_{i+1} - M$. In either case, the claim is satisfied. The proof for $p(T_{i+1})$ is identical.

Using this relation, we have the algorithm:

```
Schedule(B, W, M, n) {
    X[1] = B[1]
    Y[1] = W[1] - M

    for (i from 2 to n) {
        X[i] = B[i] + max(X[i-1], Y[i-1] - M)
        Y[i] = W[i] + max(X[i-1] - M, Y[i-1])
    }

    return max(X[n], Y[n])
}
```

For running time, each iteration of the loop takes $O(1)$ time, so the total time is $O(n)$. It is correct by the argument above. □

3 Word Segmentation (33 points)

A number of languages, both ancient and modern, are written without spaces between the words. In these languages, an important problem is word segmentation: given a string of characters, divide the string into consecutive words. In English, the analogous problem would consist of taking a string like "meetateight" and deciding that the best segmentation is "meet at eight" (and not "me et at eight" or "meet ate ight", etc.).

A natural approach to this problem is to find a segmentation that maximizes the cumulative "quality" of its individual constituent words. Thus, suppose you are given a black box that, for any string of letters $x = x_1x_2x_3 \dots x_k$, will return a number $quality(x)$. This number can be either positive or negative; larger numbers correspond to more plausible words. (So $quality($"me"$)$ would be positive, while $quality($"ght"$)$ would be negative.)

Given a long string of letters $y = y_1y_2 \dots y_n$, a segmentation of y is a partition of its letters into contiguous blocks of letters; each block corresponds to a word in the segmentation. The *total quality* of a segmentation is the sum of the qualities of the blocks.

Give an algorithm that takes a string y and computes the maximum total quality of any segmentation (i.e., return the largest number α such that there is a segmentation of total quality α and no segmentation has total quality larger than α). Your algorithm should run in $O(n^2)$ time. As always, prove correctness and running time. You can treat a single call to the black box computing $quality(x)$ as a single computational step.

Proof. Let $Y_i = y_1y_2 \cdots y_i$ and let $X_i = \{j_1, j_2, \cdots, j_m, i\}$ be the optimal segmentation for Y_i, where the words end on y_i and y_{j_k} for $k = 1, \cdots, m$, and j_m is maximal. Let $q(S)$ be the quality associated with a segmentation S. We claim that $X_i \setminus \{i\}$ is the optimal segmentation for Y_{j_m}.

Suppose otherwise. Since X_{j_m} is optimal, we have $q(X_{j_m}) > q(X_i \setminus \{i\})$. Then let $\hat{X}_i = X_{j_m} \cup \{i\}$. Now,

$$\begin{aligned} q(\hat{X}_i) &= q(X_{j_m}) + quality(y_{j_m+1}y_{j_m+2} \cdots y_i) \\ &> q(X_i \setminus \{i\}) + quality(y_{j_m+1}y_{j_m+2} \cdots y_i) \\ &= q(X_i) \end{aligned}$$

which is a contradiction since X_i was assumed to be the optimal segmentation. Now, since the choice of j_m must maximize $q(X_i)$, we have

$$q(X_i) = \max_{1 \le j_m \le i} \{q(X_{j_m}) + quality(y_{j_m+1}y_{j_m+2} \cdots y_i\} \tag{2}$$

Thus, to find the optimal segmentation, we have the algorithm

```
Segment(string, n) {
    M[0] = 0

    for (i from 1 to n) {
        M[i] = max(M[j] + quality(string[j+1:i])
    }
}
```

The total number of computations the max function computes over all loops is

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2} = O(n^2)$$

so the running time is $O(n^2)$, as desired. To prove correctness of $M[n]$, proceed by induction. Let $OPT(i)$ be the quality of the optimal segmentation of Y_i. The base case is obvious, since $M[0] = 0$ because the string is

empty. Now suppose $M[k] = OPT(k)$ for some $1 \leq k < n$. Then we have

$$\begin{aligned} M[k+1] &= \max_{1 \leq j \leq k+1} \{M[j] + quality(y_j y_{j+1} \cdots y_{k+1}\} \\ &= \max_{1 \leq j \leq k+1} \{OPT(j) + quality(y_j y_{j+1} \cdots y_i\} \\ &= OPT(k+1) \end{aligned}$$

from (2) as desired. Thus, $M[n]$ is the quality of the optimal segmentation of y. □