CSC373 - Problem Set 3

Authors: Luke Bacchus, Naslin Rahman, Zhuoqian Li

February 16, 2021

Question 1

a. Let A(i, h) = maximum grade one can receive on i projects if they spend h number of hours on it. The Bellman equation is as follows. For each value of k between 0 and h we calculate the max value of k that maximizes $f_i(k)$ and A[i-1, H-k] which is the max grade one can receive with the remaining time and projects.

$$A(i,h) = \begin{cases} 0, & \text{if } j = 0. \\ max_{0 \le k \le H}(A[i-1, H-k] + f_i(k)), & \text{otherwise.} \end{cases} \tag{1}$$

To maximize the average grade over n courses we will do max $\frac{(f_1(h1)+f_2(h2)...+f_n(hn))}{n}$ which is equivalent to maximizing $\sum_{i=1}^{n} f_i(hi)$

```
BottomUp(n, H):
    for h = 0 to H:
        A(0, h) = 0
    for i = 1 to n:
        A(i, 0) = f_i(0)

    for i = 1 to n:
        max = 0
        for j = 0 to H:
            for k = 0 to j
                g = A(i-1, H-k) + f_i(k)
                if (g > max):
                    max = g
            A(i, j) = max
    return A(n, H) /n
```

The running time is $O(H + n + nH^2)$. The inner two loop takes $O(H^2)$ time and we run this block of code n times. Total runtime $O(nH^2)$

b.

```
Augmented(n, H):
    for h = 0 to H:
        A(0, h) = 0
    for i = 1 to n:
        A(i, 0) = f_i(0)

    for i = 1 to n:
        max = 0
        for j = 0 to H:
            for k = 0 to j
                g = A(i-1, H-k) + f_i(k)
                if (g > max):
                    max = g
                    hours[i] = k
            A(i, j) = max
    return hours and A(n, H) /n
```

This question was written and read by Luke, Naslin, and Zhuoqian.

Question 2

a. The subproblem is breaking down the question into smaller sections, and finding the most optimal way of connecting those smaller sections. Then by finding the most optimal way to connect those sections. This works because we are also trying out every way in which these sections can be divided and thus are trying every possible way of connecting the carts.

The subproblem of this question is defined as: cost(i,j): The cost of connecting train carts i to j. The Bellman equation is as follows:

$$cost(i,j) = \begin{cases} 0, & \text{if } i = j. \\ min_{i \le k < j}(cost(i,k) + cost(k+1,j) + & \\ min((w_i + ... + w_k)^{1/2}, (w_{k+1} + ... + w_j)^{1/2})), & \text{if } i < j. \end{cases} \quad (2)$$

Pseudocode:

```
BottomUp(n, W):
    for i = 1 to n:
        cost(i, i) = 0

    for l = 1 to n-1:
        for i = 1 to n-1:
            j = i + l
            if j <= n:
                min_found = inf
                for k = i to j-1:
                    found = min(cost(i,k) + cost(k+1,j) +
                    min(w[i:k]^1/2, w[k+1:j]^1/2))

                    if found < min_found:
                        min_found = found

            cost(i,j) = min_found
    return cost(1,n)
```

Time Complexity: O(n^3) since we have 3 nested for loops

b. To augment the algorithm so that it also outputs an optimal order of train car connections, we can add to the memory part of the algorithmn by adding another array that stores the order. The order is an array that stores pairs in the order in which the train carts are connected. Ex: [[1,2],[3,4],[5,6],[2,3],[4,5]]

Pseudocode:

```
BottomUp(n, W):
    for i = 1 to n:
        cost(i, i) = 0
        order(i,i) = []

    for l = 1 to n-1:
        for i = 1 to n-1:
            j = i + l
            if j <= n:
                min_found = inf
                for k = i to j-1:
                    found = min(cost(i,k) + cost(k+1,j) +
                    min(w[i:k]^1/2, w[k+1:j]^1/2))

                    found_order = order(i,k) + order(k+1,j) + [[k,k+1]]

                    if found < min_found:
                        min_found = found
                        min_order = found_order

            cost(i,j) = min_found
            order(i,j) = min_order
    return cost(1,n)
```

This question was written and read by Luke, Naslin, and Zhuoqian.

Question 3

a. Let y = Green
Let x = El
Let z = Greelen

The greedy algorithmn would take the 'Gree' portion of z and remove that from y such that y = en. What remains is x = el, y = en, z = len, and from there, the greedy algorithm cannot continue further and must return false.

b. The subproblem of this question is defined as IsInterleaving(i, j): True iff $z[1]...z[i+j]$ is an interleaving of $x[1]...x[i]$ and $y[1]...y[j]$. The Bellman equation is as follows:

$$\text{interleaving}(i,j) = \begin{cases} \text{True}, & \text{if } i = 0 \text{ and } j = 1 \\ & \text{and } z[1] = y[1] \\ & \text{OR} \\ & \text{if } i = 1 \text{ and } j = 0 \\ & \text{and } z[1] = x[1]. \\ \\ z[i+j] == x[i] \text{ and interleaving}(i-1, j) & \\ \text{OR} & \text{otherwise.} \\ z[i+j] == y[j] \text{ and interleaving}(i, j-1) & \end{cases}$$

Let the length of $x = n$ and the length of $y = m$. Pseudocode:

```
IsInterleaving(x, y, z)
    n = |x|
    m = |y|

    In is 2dArray [i, j] of Bools.
        for i = 1 to n
        and for j = 1 to m

    Initialize In to False at every index

    In[0, 1] = z[1] == y[1]
    In[1, 0] = z[1] == x[1]

    for i = 1 to n
        for j = 1 to m
            In[i, j] = (z[i+j] == x[i] and In[i-1, j]) or \
```

```
                (z[i+j] == y[j] and In[i, j-1])
    return In[n, m]
```

The running time of this program is $\mathcal{O}(nm)$, as initializing the array takes nm time, and the second loop also takes nm time. This algorithm is correct because it recursively creates a 2D array of Bools based on previous values and then indexes the last element of that array (length of x, length of y) after its creation.

This question was written and read by Luke, Naslin, and Zhuoqian.