HW 6

Due on **Wednesday, March 6, 2019 at 10am**

CS/ECE 374: Algorithms & Models of Computation, Spring 2019 Version: **1.1**

Submission instructions as in previous homeworks.

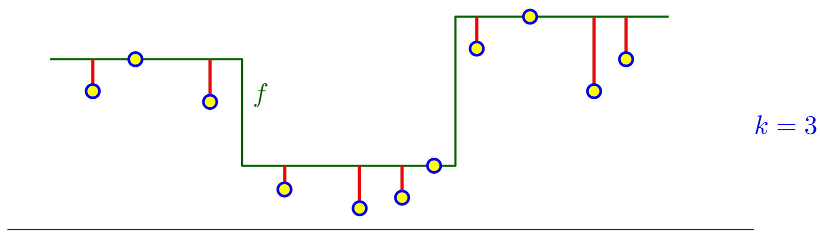
16 (100 PTS.) Simplifying data.

A k-*step function* is a function of the form

$$f(x) = b_i \qquad \text{if } a_i \leq x < a_{i+1} \quad (i = 0, \ldots, k-1)$$

for some $-\infty = a_0 < a_1 < \cdots < a_{k-1} < a_k = \infty$ and some $b_0, b_1, \ldots, b_{k-1}$.

We are given n data points $p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)$ and a number k between 1 and n. Our objective is to find a k-step function f such that $f(x_i) \geq y_i$ for all $i \in \{1, \ldots, n\}$, while minimizing the total "error" $\sum_{i=1}^{n}(f(x_i) - y_i)$ (this is the total length of the red vertical segments in the figure below).



16.A. (70 PTS.) Describe an algorithm, as fast as possible, that computes the minimum total error of the optimal k-step function. Bound the running time of your algorithm as a function of n and k.

[Note: in dynamic programming questions such as this, first give a clear English description of the function you are trying to evaluate, and how to call your function to get the final answer, then provide a recursive formula for evaluating the function (including base cases). If a correct evaluation order is specified clearly, iterative pseudocode is not required.]

Solution:

First sort the points so that $x_1 \leq x_2 \leq \cdots \leq x_n$. (Set $x_0 = -\infty$.) This step takes $O(n \log n)$ time using merge sort, for example.

For each $i \in \{0, \ldots, n\}$ and $\ell \in \{0, \ldots, k\}$, define $E(i, \ell)$ to be the total error of the optimal ℓ-step function for the points $p_1, \ldots, p_i$.

The final answer we want is $E(n, k)$.

Base cases: $E(0, 0) = 0$ and $E(i, 0) = \infty$ for all $i \in \{1, \ldots, n\}$. (We could also include $E(0, \ell) = \infty$ for all $\ell \in \{1, \ldots, k\}$.)

Recursive formula: for $\ell > 0$,

$$E(i, \ell) \;=\; \min_{j=1,\ldots,i} \left(E(j-1, \ell-1) + (j-i+1)m(j,i) - s(j,i) \right)$$

where

$$m(j, i) = \max\{y_j, \ldots, y_i\}, \qquad s(j, i) = y_j + \cdots + y_i.$$

Explanation: Here, j corresponds to the index for which the last "step" of the optimal step function covers points $p_j, \ldots, p_i$. The last step would have y-value $m(j,i)$ and the error contributed is precisely $(j-i+1)m(j,i) - s(j,i)$. We don't know the right choice of j ahead of time, so we try all of them and take the minimum.

We can first compute each $m(j,i)$ and $s(j,i)$ in $O(j-i+1) \leq O(n)$ time, and store all $O(n^2)$ values in a table. The total time so far is $O(n^3)$.

We can then evaluate $E(\cdot, \cdot)$ by memoization, or bottom-up in increasing order of ℓ (or increasing order of i) using a table. There are $O(kn)$ subproblems, each requiring $O(n)$ time, giving a total of $O(kn^2)$ time.

The overall running time is $O(n^3 + kn^2) = O(n^3)$.

Improvement. The precomputation of all the $m(j,i)$ and $s(j,i)$ values can be done in $O(n^2)$ time with more care. Instead of computing each value from scratch, observe that $m(j,i) = \max\{m(j,i-1), y_i\}$ and $s(j,i) = s(j,i-1) + y_i$. As a result, the overall running time can be reduced to $O(kn^2)$.

What does not work. It is tempting to try and compute $E(i,\ell)$ by thinking about adding the ith point to the solution from $E(i-1,\ell)$ by just adding the new point to the last step (you might have to adjust the height of the step), or alternatively, start a new step, and then use $E(i-1,\ell-1)$ as the value. The problem with this approach is that adding the ith point to the last stair might make it much higher. But then, the resulting solution for the prefix of $i-1$ points is no longer optimal, Namely, it is critical for the prefix optimal property to hold that we guess where the stairs start and end.

16.B. (30 PTS.) Describe how to modify your algorithm in (A) so that it computes the optimal k-step function itself.

Solution:

For each $i \in \{0, \ldots, n\}$ and $\ell \in \{1, \ldots, k\}$, let pred$[i,\ell]$ be the index j that minimizes $E(j-1,\ell-1) + (j-i+1)m(j,i) - s(j,i)$. We can compute all pred$[i,\ell]$ values while we are computing $E(i,\ell)$, without increasing the asymptotic running time. Afterwards, we can generate the optimal step function by calling **OutputStepFunction**(n,k):

```
OutputStepFunction(i, ℓ):
  if ℓ = 0 then return
  j = pred[i, ℓ]
  OutputStepFunction(j − 1, ℓ − 1)
  output that the step function has y-value m(j, i) for x_{j−1} < x ≤ x_i
```

This takes $O(k)$ additional time.

17 (100 PTS.) Closest subsequence

Define the L_1-*distance* between two sequences of real numbers $\langle a_1, \ldots, a_m \rangle$ and $\langle b_1, \ldots, b_m \rangle$ to be $|a_1 - b_1| + \cdots + |a_m - b_m|$.

Consider the following problem: given two sequences of real numbers $A = \langle a_1, \dots, a_m \rangle$ and $B = \langle b_1, \dots, b_n \rangle$ with $m \le n$, find a subsequence of B of length m that minimizes its L_1-distance to A.

17.A. (70 PTS.) Describe an algorithm, as fast as possible, that computes the L_1-distance of the optimal subsequence of B to A. Bound the running time of your algorithm as a function of m and n.

Solution:

For each $i \in \{0, \dots, m\}$ and $j \in \{0, \dots, n\}$, define $D(i, j)$ to be the L_1-distance of the closest subsequence of $\langle b_1, \dots, b_j \rangle$ to $\langle a_1, \dots, a_i \rangle$.

The final answer we want is $D(m, n)$.

Base cases: $D(0, 0) = 0$ and $D(i, 0) = \infty$ for all $i \in \{1, \dots, m\}$.

Recursive formula:

$$D(i, j) = \min\{D(i, j-1),\ D(i-1, j-1) + |a_i - b_j|\}.$$

Explanation: The first term corresponds to the case when the optimal solution for $D(i, j)$ does not use b_j, and the second term corresponds to the case when the optimal solution uses b_j to match with a_i.

We can evaluate $D(\cdot, \cdot)$ by memoization, or bottom-up in increasing order of j using a table. There are $O(mn)$ subproblems, each requiring $O(1)$ time, giving a total of $O(mn)$ time.

17.B. (30 PTS.) Describe how to modify your algorithm in (A) so that it computes the optimal subsequence itself.

Solution:

We call **OutputSubsequence**(m, n):

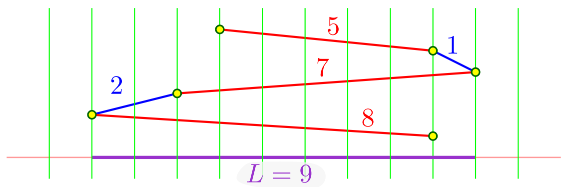
```
OutputSubsequence(i, j):
   if j = 0 then return
   if D(i, j) = D(i, j − 1) then
      OutputSubsequence(i, j − 1)
   else
      OutputSubsequence(i − 1, j − 1)
      output b_j
```

This takes $O(n)$ additional time.

18 (100 PTS.) Fold it!

We are given a "chain" with n links of lengths $a_1, \ldots, a_n$, where each a_i is a positive integer. We are also given a positive integer L. We want to determine if it is possible to "fold" the chain (in one dimension) so that the length of the folded chain is at most L. More formally, we want to decide whether there exists $t \in [0, L]$ and $s_1, \ldots, s_n \in \{-1, +1\}$ such that $t + \sum_{i=1}^{j} s_i a_i \in [0, L]$ for all $j \in \{0, \ldots, n\}$. (Here, t denotes the starting position, and $s_i = \pm 1$ depending on whether we turn rightward or leftward for the ith link.)

Example: for $a_1 = 5$, $a_2 = 1$, $a_3 = 7$, $a_4 = 2$, $a_5 = 8$, and $L = 9$, a solution is shown below.



18.A. (70 PTS.) Provide an $O(nL)$-time algorithm to decide whether a solution exists. (Argue why the stated running time is correct.)

Partial credit would be given to slower solutions with running time $O(nL^2)$ or $O(nL^3)$.

Solution:

For each $i \in \{0, \ldots, n\}$ and $j \in \{0, \ldots, L\}$, define $A(i, j)$ to be true iff there is a way to fold the chain $\langle a_1, \ldots, a_i \rangle$ so that all vertices lie in $[0, L]$ and the last vertex is at position j.

The final answer is $\bigvee_{j=0}^{L} A(n, j)$.

Base cases: $A(0, j) =$ true for all $j \in \{0, \ldots, L\}$.

Recursive formula:

$$A(i,j) = \begin{cases} A(i-1, j-a_i) \vee A(i-1, j+a_i) & \text{if } a_i \le j \le L - a_i \\ A(i-1, j-a_i) & \text{if } j \ge a_i \text{ and } j > L - a_i \\ A(i-1, j+a_i) & \text{if } j \le L - a_i \text{ and } j < a_i \\ \text{false} & \text{else} \end{cases}$$

Explanation: The $A(i-1, j-a_i)$ term corresponds to the case when the last link in the optimal solution is a right turn. The $A(i-1, j+a_i)$ term corresponds to the case when the last link in the optimal solution is a left turn. We don't know which choice to take beforehand, so will try both (unless indices are out-of-bound) and take the "or" of the two options.

We can evaluate $A(\cdot, \cdot)$ by memoization, or bottom-up in increasing order of i using a table. There are $O(nL)$ subproblems, each requiring $O(1)$ time, giving a total of $O(nL)$ time.

18.B. (30 PTS.) Using (A) as a subroutine, describe an algorithm (as fast as possible) to find the minimum length L^* such that a valid folding exists. What is the running time of your algorithm as a function of n and L^*?

Solution:

Part (A) gives us a procedure **decide**(L) that returns yes iff the minimum length L^* is at most L, in $O(nL)$ time.

Naive solution. We call **decide**(L) for $L = 1, 2, 3, \ldots$ until it returns yes. This requires L^* calls to **decide**$(\cdot)$, for a total time of $O(n(L^*)^2)$.

Better solution.

> **for** $k = 1, 2, \ldots$
>   **if** **decide**(2^k) returns true then break
> do binary search to find the smallest $L \in [2^{k-1}, 2^k]$ such that
>   **decide**(L) returns true

The first two lines take $O(\sum_{k=1}^{\lceil \log L^* \rceil} n2^k) = O(n2^{\lceil \log L^* \rceil + 1}) = O(nL^*)$ time. The final binary search requires $O(\log L^*)$ iterations and takes $O(nL^* \log L^*)$ time. The overall running time is $O(nL^* \log L^*)$.

Another solution. Let $L_{\max} = \max\{a_1, \ldots, a_n\}$. Observe that $L^* \leq nL_{\max}$, and $L^* \geq L_{\max}$. We can do binary search in the range $[L_{\max}, nL_{\max}]$, requiring $O(\log(nL_{\max})) \leq O(\log(nL^*))$ iterations. The overall running time of this solution is $O(nL^* \log(nL^*))$ time, which is close enough. (In fact, a more careful argument would show that $L^* \leq 2L_{\max}$, and so we can again get $O(nL^* \log L^*)$ time.)