CS381 Homework 2 Problem 1

Connor Couetil

February 5, 2026

1 Exercise 5.1.2

`convex-subsequence(`i`)`: Given an array $A[1, ..., n]$ as global and parameter $i \in [n]$, returns the length of the longest convex subsequence in $A[1, ..., i]$.

1. if $i \leq 2$, Return 0
2. Return $Max(\{1 + \texttt{convex-subsequence}(j) : j, k \in [n], 2 \leq j < i - 1, 1 \leq k < j - 1, A[i] - A[j] \geq A[j] - A[k]\} \cup \{\texttt{convex-subsequence}(i - 1)\})$

To produce the length of the longest convex subsequence in $A[1, ..., n]$ call `convex-subsequence(`n`)`. The function uses three indexes to track all possible pairs of line segments that share a node. The search space is n^3 for an $O(n^3)$ runtime.

2 Exercise 5.1.4

`even(`i`)`: Given $A[1, ..., n]$ as a global and $i \in [n]$, returns the length of the longest subsequence of $A[1, ..., i]$ which has even sum.

1. If i is 1
   A. If $A[i]$ is even, Return 1
   B. If $A[i]$ is odd, Return 0
2. If $A[i]$ is even
   A. Return $Max(\{\texttt{even}(i - 1)\} \cup \{\texttt{even}(j) + 1$ for $1 \leq j < i$ where $A[j] < A[i]\})$
3. If $A[i]$ is odd
   A. Return $Max(\{\texttt{even}(i - 1)\} \cup \{\texttt{odd}(j) + 1$ for $1 \leq j < i$ where $A[j] < A[i]\})$

`odd(`i`)`: Given $A[1, ..., n]$ as a global and $i \in [n]$, returns the length of the longest subsequence of $A[1, ..., i]$ which has odd sum.

1. If i is 1
   A. If $A[i]$ is even, Return 0
   B. If $A[i]$ is odd, Return 1
2. If $A[i]$ is even
   A. Return $Max(\{\texttt{odd}(i - 1)\} \cup \{\texttt{odd}(j) + 1 : 1 \leq j < i, A[j] < A[i]\})$
3. If $A[i]$ is odd

A. Return $Max(\{\texttt{odd}(i-1)\} \cup \{\texttt{even}(j)+1 : 1 \leq j < i,\ A[j] < A[i]\})$

To produce the largest subsequence that sums to an odd or even number for an array A of size n, simply call `odd(`n`)` or `even(`n`)`, respectively. The runtime of the above functions is $O(n^2)$, it will go through all combinations of indices representing the last two values in the subsequence, of which there are $\frac{n(n-1)}{2}$. Repeated calls to the above functions are memoized, reducing the work performed to the parameter space of the functions.

3 Exercise 5.1.6

<u>`american(`i`)`</u>: Given $A[1,...,n]$ as a global and $i \in [n]$, returns the length of the longest subsequence of $A[1,...,i]$ which has alternating colors red, white, blue, ... whose start is any of the colors.

1. If i is 1, Return 1
2. Return $Max(\{\texttt{american}(i-1)\} \cup \{\texttt{american}(j)+1 : 1 \leq j < i, A[i]$ is next color after $A[j]\})$

To produce the largest subsequence of american colors in A of length n, simply call `american(`n`)`. Repeated calls to the function are cached based on the parameter i. The runtime is $O(n)$, there are n possible total different calls to the function even though the total number of function calls made is quadratic in the worst case.

4 Exercise 5.1.7

<u>`palindrome(`i`, `j`)`</u>: Given $A[1,...,n]$ as a global and $i, j \in [n]$, returns the length of the longest subsequence of in A that is a palindrome.

1. If $j < i$ Return 0
2. If $j = i$ Return 1
3. If $A[j] = A[i]$ Return $2 + \texttt{palindrome}(i+1,\ j-1)$
4. Return Max($\texttt{palindrome}(i+1,\ j)$, $\texttt{palindrome}(i,\ j-1)$)

To get the length of the longest palindromic subsequence of a string A of length n, call `palindrome(1, `n`)`. The runtime of this algorithm is $O(n^2)$ when the function is memoized based on parameters i and j. There are two recursive calls that sweep right and left, respectively, producing all combinations of i and j in $[n]$ where $i < j$. Constant work is performed within each call of the function.