CS/ECE 374 Spring 2025 Heng An Cheng (hacheng2@illinois.edu)
Homework 7 Problem 1

Solution:

(a) **Idea**

Sort points by x-coordinate in $O(n \log n)$ time. Scan from right to left, keeping a point if its y-coordinate is greater than the maximum y seen so far.

Why It Works

A point $p_i = (x_i, y_i)$ is undominated if no other point has both a larger x and a larger y. After sorting by x, scanning right to left lets us track the maximum y of points with bigger x-coordinates. If y_i exceeds this maximum, p_i isn't dominated by any point to its right, and points to its left have smaller x, so they can't dominate it either. This runs in $O(n \log n)$ time.

Algorithm 1 Find Undominated Points (Sorting and Scanning)

procedure FIND_UNDOMINATED(P, n)
  $P \leftarrow$ sort$(P, \text{key} = x)$ ▷ Sort by x-coordinate ascending, $O(n \log n)$
  $Q \leftarrow []$ ▷ Initialize empty list for undominated points
  max_y $\leftarrow 0$ ▷ Track maximum y-coordinate seen
  for $i = n - 1$ **downto** 0 **do** ▷ Scan from highest x to lowest
    if $P[i].y >$ max_y **then** ▷ Check if point is undominated
      Q.add$(P[i])$ ▷ Add to undominated set
      max_y $\leftarrow P[i].y$ ▷ Update maximum y
    end if
  end for
  return reverse(Q) ▷ Optional: return in increasing x order
end procedure

Time Analysis

- Sorting the points by their x-coordinates in ascending order takes $O(n \log n)$ time.
- Scanning from the largest x-coordinate to the smallest takes $O(n)$ time.
- For each point, checking whether it is undominated takes $O(1)$ time by maintaining the current maximal y-coordinate.

Therefore, the total time complexity is

$$O(n \log n)$$

(b) **Argue that for any k there is always a subset of Q that is an optimum solution.**

For any integer k, suppose there exists an optimal solution of size k that includes a point $A \notin Q$. Since A is dominated, there exists a point $B \in Q$ that dominates A. Replacing A with B does not decrease the number of dominated points, as B dominates at least all points that A does.

Thus, for any k, any optimal solution can be transformed into one consisting only of points from Q. □

Idea

We do preprocessing in advanced

- Compute Q from the method discussed in (a):

$$O(n \log n)$$

- Precompute $D(q_i)$, the set of points dominated by each $q_i \in Q$:

$$O(nm) \text{ total, where } m = |Q| \leq n$$

We define :

$dp[i][k] =$ the maximum number of points in P dominated by a subset of $\{q_0, q_1, \ldots, q_i\}$ with exactly k points.

$S[i][k] =$ set of dominated points acheiving dp[i][k]

Transitions:

- **Do not choose** q_i:

$$dp[i][k] = dp[i-1][k], \quad S[i][k] = S[i-1][k]$$

- **Choose** q_i: Let $D(q_i)$ be the set of points dominated by q_i.

$$dp[i][k] = |S[i-1][k-1] \cup D(q_i)|, \quad S[i][k] = S[i-1][k-1] \cup D(q_i)$$

At the end, we take:

$$\max dp[m][k]$$

where $m = |Q|$.

Algorithm 2 Dynamic Programming for Maximum Dominated Points (Non-Y-Rank)

Input: Set of points P, integer k
Sort P by decreasing x-coordinate: $p_1, p_2, \ldots, p_n$ ▷ $O(n \log n)$
Compute $Q = \{q_0, q_1, \ldots, q_{m-1}\}$ (maximal points) ▷ $O(n \log n)$
for $i = 0$ to $m - 1$ **do**
    $D(q_i) = \{p \in P \mid p_x \leq q_{i_x}, p_y \leq q_{i_y}\}$ ▷ Precompute, $O(nm)$ total
end for
Initialize $\text{dp}[0][0] = 0$, $S[0][0] = \emptyset$
$\text{dp}[0][1] = |D(q_0)|$, $S[0][1] = D(q_0)$
for $k' = 2$ to k **do**
    $\text{dp}[0][k'] = -\infty$, $S[0][k'] = \emptyset$
end for
for $i = 1$ to $m - 1$ **do**
    for $k' = 0$ to k **do**
        $\text{dp}[i][k'] = \text{dp}[i-1][k']$ ▷ Don't choose q_i
        $S[i][k'] = S[i-1][k']$
        if $k' \geq 1$ **then**
            $S_{\text{new}} = S[i-1][k'-1] \cup D(q_i)$ ▷ Union in $O(n)$
            $\text{val} = |S_{\text{new}}|$
            if $\text{val} > \text{dp}[i][k']$ **then**
                $\text{dp}[i][k'] = \text{val}$
                $S[i][k'] = S_{\text{new}}$
            end if
        end if
    end for
end for
Return: $\max_{k'=0}^{k} \text{dp}[m-1][k']$

Time Analysis

The total time complexity is composed of the following steps:

- **Compute** Q: $O(n \log n)$.
- **Precompute** $D(q_i)$: For each $q_i \in Q$ ($m = |Q| \leq n$), scan all n points to find dominated points in $O(nm)$ total.
- **Dynamic Programming**: For each of the $O(mk)$ states, union sets in $O(n)$ time, totaling $O(mkn)$. With $m, k \leq n$, this is $O(n^3)$.

Total Time Complexity:

$$O(n \log n) + O(n^2) + O(n^3) = O(n^3)$$

CS/ECE 374 Spring 2025 Heng An Cheng (hacheng2@illinois.edu)

Homework 7 Problem 2

Solution:

(a) Let LIPS(i, j) denote the length and Lastchar denote the first (or the last) character of the longest interesting palindrome subsequence of A[i .. j]. Use simliar idea from Lab, but add condition $A[j] \neq Lastchar(i+1, j-1)$ to prevent adjacent characters in w are the same.

This function obeys the following recurrence:

$$(LIPS(i,j),\, Lastchar(i,j)) = \begin{cases} (0,\, \phi) & \text{if } i > j, \\ (1,\, \phi) & \text{if } i = j, \\ (2 + LIPS(i+1, j-1),\, A[i]) & \text{if } i < j,\ A[i] = A[j], \\ & \text{and } A[j] \neq Lastchar(i+1, j-1), \\ \begin{cases} (LIPS(i+1,j),\, Lastchar(i+1,j)) \text{ if } LIPS(i+1,j) \geq LIPS(i,j-1) \\ (LIPS(i,j-1),\, Lastchar(i,j-1)) \text{ if } LIPS(i+1,j) < LIPS(i,j-1) \end{cases} \end{cases}$$

Algorithm 3 Compute LIPS (Longest Interesting Palindromic Subsequence) Table

for $i = n$ down to 1 **do**
  $LIPS(i, i-1) \leftarrow 0$
  $LIPS(i, i) \leftarrow 1$
  $Lastchar(i, i) \leftarrow A[i]$
  for $j = i + 1$ to n **do**
    if $A[i] = A[j]$ **and** $A[j] \neq Lastchar(i+1, j-1)$ **then**
      $LIPS(i, j) \leftarrow 2 + LIPS(i+1, j-1)$
      $Lastchar(i, j) \leftarrow A[i]$
    else
      if $LIPS(i+1, j) \geq LIPS(i, j-1)$ **then**
        $LIPS(i, j) \leftarrow LIPS(i+1, j)$
        $Lastchar(i, j) \leftarrow Lastchar(i+1, j)$
      else
        $LIPS(i, j) \leftarrow LIPS(i, j-1)$
        $Lastchar(i, j) \leftarrow Lastchar(i, j-1)$
      end if
    end if
  end for
end for

Time Analysis

The algorithm has two nested loops, each running up to n times, resulting in $O(n^2)$ iterations. Each operation inside the loops takes constant time, so the overall time complexity is:

$$O(n^2)$$

(b) Let LCPS(i, j, x, y) denote the Longest Common Palindromic Subsequence (LCPS) for the substring $a[i..j]$ of string a and $b[x..y]$ of string b.

This function obeys the following recurrence:

$$\text{LCPS}(i,j,x,y) = \begin{cases} 0 & \text{if } i > j \text{ or } x > y, \\ 1 & \text{if } i = j \text{ and } x = y \text{ and } a[i] = b[x], \\ 0 & \text{if } i = j \text{ and } x = y \text{ and } a[i] \neq b[x], \\ 2 + \text{LCPS}(i+1, j-1, x+1, y-1) \quad \text{if } a[i] = a[j] = b[x] = b[y], \\ \begin{cases} \max\left(\text{LCPS}(i+1,j,x,y), \text{LCPS}(i,j-1,x,y)\right) & \text{if } a[i] \neq b[x] \text{ or } a[j] \neq b[y], \\ \max\left(\text{LCPS}(i,j,x+1,y), \text{LCPS}(i,j,x,y-1)\right) & \text{otherwise.} \end{cases} \end{cases}$$

Algorithm 4 Longest Common Palindromic Subsequence (LCPS)

```
function LCPS(a, b, i, j, x, y)
    if i > j or x > y then
        return 0
    end if
    if i = j and x = y and a[i] = b[x] then
        return 1
    end if
    if a[i] = a[j] = b[x] = b[y] then
        return 2+ LCPS(a, b, i+1, j-1, x+1, y-1)
    else
        return max(LCPS(a, b, i+1, j, x, y), LCPS(a, b, i, j-1, x, y), LCPS(a, b, i, j, x+1,
y), LCPS(a, b, i, j, x, y-1))
    end if
end function
```

Time Analysis

The recurrence LCPS(i, j, x, y) involves four indices: i, j for string a, and x, y for string b. Each index ranges from 1 to n, so the total number of possible subproblems is $O(n^4)$.

Since each subproblem is solved in constant time $O(1)$ and there are $O(n^4)$ distinct subproblems, the overall time complexity is:

$$O(n^4)$$

■