Assignment 9

Sample Solutions

CSCI 3110 — Fall 2018

(a) Let us denote the sequence of dominoes by $S = \langle d_1, d_2, \dots, d_n \rangle$; each domino d_i is a pair $[x_i : y_i]$. First, let us try to come up with a recurrence for the length of a longest domino subsequence (LDS) of S. Let $\ell(i)$ denote the longest domino subsequence of S that ends in domino d_i, and let L be the length of the LDS of S. Then, obviously, since the LDS has to end in some domino, we have

$$L = \max_{1 \le i \le n} \ell(i).$$

So we have to compute only the values $\ell(1), \ell(2), \dots, \ell(n)$. Let S_i be the LDS of S that ends in domino d_i. If $y_j \ne x_i$, for all $1 \le j < i$, then S_i must have length one because there is no domino that can precede d_i in S_i. Otherwise, the domino d_j that precedes d_i in S_i must satisfy $y_j = x_i$.

Next observe that, for the domino d_j that precedes d_i in S_i, the prefix of S_i that ends in d_j must be S_j (or a domino sequence ending in d_j of equal length). Indeed, if this was not the case, we could construct a longer domino subsequence than S_i that ends in d_i: Take S_j, which ends in d_j, and append d_i. So, now the structure of S_i is clear: It consists of a longest domino sequence S_j that ends in the predecessor d_j of d_i in S_i, followed by d_i.

How do we choose the predecessor? Well, out of all dominoes d_j with $y_j = x_i$, we obviously want to choose the one whose sequence S_j has maximal length. This gives the following recurrence:

$$\ell(i) = 1 + \max(\{0\} \cup \{\ell(j) \mid 1 \le j < i \text{ and } y_j = x_i\}$$

Based on this recurrence, we can now compute an LDS of S using the following algorithm. The algorithm constructs two tables ℓ and L such that $\ell[i]$ stores the length of an LDS that ends in d_i an $L[i]$ stores such an LDS with the dominoes listed back to front, represented as a singly linked list. As we did for other problems in class, many of the sequences $L[1], \dots, L[n]$ share most of their representation. The input of the algorithm is the sequence S of dominoes. The x- and y-fields of the ith domino are accessed as $S[i].x$ and $S[i].y$.

```
LDS-DP(S)
for i ← 1 to |S|
     do ℓ[i] ← 1
        L[i] ← ⟨S[i]⟩
        for j ← 1 to i − 1
            do if S[j].y = S[i].x and ℓ[j] + 1 > ℓ[i]
                  then ℓ[i] ← ℓ[j] + 1
                       L[i] ← ⟨S[i]⟩ ∘ L[j]
return (ℓ, L)
```

Given the output of LDS-DP, the final LDS can now be found using the following wrapper:

```
LDS(S)
(ℓ, L) ← LDS-DP(S)
m ← 1
for i ← 2 to |S|
     do if ℓ[i] > ℓ[m]
           then m ← i
return REVERSE(L[m])
```

The correctness of this solution follows from the discussion we used to derive the recurrence for $\ell(i)$. The running time is $O(n^2)$. Clearly, lines 2–5 of procedure LDS take $O(n)$ time. Since the sequence stored in $L[m]$, line 6 also takes $O(n)$ time. Thus, we only need to argue that procedure LDS-DP, invoked in line 1 of procedure LDS takes $O(n^2)$ time. Procedure LDS-DP consists of a for-loop with n iterations in lines 1–7. Each iteration of this loop takes constant time plus up to n iterations of the loop in lines 5–7, at a constant cost per iteration. Thus, procedure LDS-DP takes $O(n^2)$ time.

(b) Now observe that procedure LDS would take $o(n^2)$ time if we could replace the loop in lines 5–7 of procedure LDS-DP with something that takes $o(n)$ time. Excluding the time spent in lines 5–7 of procedure LDS-DP, procedure LDS takes only linear time. Let us revisit the problem that lines 5–7 solve: They decide whether there exists an index $j < i$ such that $S[j].y = S[i].x$ and, among all such indices, picks the one that maximizes $\ell[j]$. Since there are only n possible values $S[i].x$, we can support this operation in constant time using a simple array m of size n. At the beginning of the ith iteration of the outer loop of procedure LDS-DP, $m[y] = 0$ if there is no index $1 \le j < i$ such that $S[j].y = y$; if there exists such a sequence, then $m[y] = j > 0$ such that $S[j].y = y$ and $\ell[j] \ge \ell[j']$ for all $1 \le j' < i$ with $S[j'].y = y$. Then $\ell[i] = 1$ if $m[S[i].x] = 0$ and $\ell[i] = \ell[m[S[i].x]] + 1$ if $m[S[i].x] > 0$. The constructed sequence ends with $S[i].y$ and as another candidate for a longest LDS that ends in $S[i].y$. Thus, we need to check whether $\ell[i] > \ell[m[S[i].y]]$; if so, we set $m[S[i].y] = i$. This gives the following faster version of procedure LDS-DP:

```
LDS-DP(S)
 1  for i ← 1 to |S|
 2       do m[i] ← 0
 3  for i ← 1 to |S|
 4       do if m[S[i].x] > 0
 5             then ℓ[i] ← ℓ[m[S[i].x]] + 1
 6                  L[i] ← ⟨S[i]⟩ ∘ L[m[S[i].x]]
 7             else ℓ[i] ← 1
 8                  L[i] ← ⟨S[i]⟩
 9          if ℓ[i] > ℓ[m[S[i].y]]
10             then m[S[i].y] ← i
11  return (ℓ, L)
```

This new version of procedure LDS-DP has two loops in lines 1–2 and in lines 3–10. Both loops have n iterations and perform a constant amount of work per iteration. Thus, procedure LDS-DP now takes $O(n)$ time, that is, the total running time of procedure LDS is $O(n)$.