Computer Science C73 — October 27, 2021
Scarborough Campus — University of Toronto

Solutions for Homework Assignment #5

Answer to Question 1.

Definition of the subproblems that our dynamic programming algorithm will solve: For each i, $0 \le i \le n$,

$$D(i) = \text{the number of legal divisions of } x[1..i] \tag{*}$$

Recursive formula to compute each subproblem:

$$D(i) = \begin{cases} 1, & \text{if } i = 0 \\ \sum_{0 \le j < i} D(j) \cdot \textsc{Dict}(x, j+1, i), & \text{if } i > 0 \end{cases} \tag{†}$$

Justification why (†) is a correct formula to compute (∗): For the base case $i = 0$, $x[1..0]$ is the empty string, which has a single legal division (vacuously, every string in the empty sequence is an English word). Therefore, $D(0) = 1$, as wanted.

For $i > 0$, consider the non-empty string $x[1..i]$. For each j such that $0 \le j < i$, if $x[j+1..i]$ is a word (i.e., $\textsc{Dict}(x, j+1, i)$ returns 1) then each of the legal divisions of $x[1..j]$, followed by $x[j+1..i]$ is a distinct legal division of $x[1..i]$. The legal divisions that arise in this way for different values of j, $0 \le j < i$, are distinct because they differ on the last string. Hence

$$D(i) \ge \sum_{0 \le j < i} D(j) \cdot \textsc{Dict}(x, j+1, i). \tag{1}$$

On the other hand, any legal division of $x[1..i]$ consists of a legal division of $x[1..j]$, for some j such that $0 \le j < i$, followed by $x[j+1..i]$, which must be an English word (otherwise the division is not legal), i.e., $\textsc{Dict}(x, j+1, i)$ returns 1. Hence

$$D(i) \le \sum_{0 \le j < i} D(j) \cdot \textsc{Dict}(x, j+1, i). \tag{2}$$

By (1) and (2), we have that, for $i > 0$, $D(i) = \sum_{0 \le j < i} D(j) \cdot \textsc{Dict}(x, j+1, i)$, as wanted.

Solving the original problem: By the definition of the subproblems (∗), the required return value is simply $D(n)$.

Pseudocode:

```
   LegalDivisions(x)
1  n := length(x)
2  D(0) := 1
3  for i := 1 to n do
4      D(i) := 0
5      for j := 0 to i − 1 do
6          D(i) := D(i) + D(j) · Dict(x, j + 1, i)
7  return D(n)
```

Running time: The running time of the algorithm is dominated by the doubly-nested loop in lines 3-6, which takes $\Theta(n^2)$ time.

Answer to Question 2.

a. <u>*Subproblems that our dynamic programming algorithm will solve:*</u> For all $i, j \in [1..n]$ such that $i \leq j$, define

$$P[i,j] = \text{minimum length of a palindrome that is a supersequence of } A[i..j] \tag{*}$$

<u>*Recursive formula to compute each subproblem:*</u>

$$P[i,j] = \begin{cases} 1, & \text{if } i = j \\ 2 + \min(P[i,j-1], P[i+1,j]), & \text{if } i < j \text{ and } A[i] \neq A[j] \\ 2, & \text{if } i = j-1 \text{ and } A[i] = A[j] \\ 2 + P[i+1,j-1], & \text{if } i < j-1 \text{ and } A[i] = A[j] \end{cases} \tag{†}$$

Note that this is recursion on $j - i$; the basis is when $j - i = 0$, i.e., $i = j$; and the general case for $P[i,j]$, when $j - i > 0$, is defined in terms of $P[i,j-1], P[i+1,j]$, and $P[i+i,j-1]$, where $(j-1)-i$, $j-(i+1)$ and $(j-1)-(i+1)$ are all smaller than $j - i$.

<u>*Justification why (†) is a correct formula to compute (∗):*</u> For the base case $i = j$ this is obvious since, in that case, $A[i..j]$ consists of a single character, so $A[i..j]$ itself is a palindrome and therefore it is its own hortest palindromic supersequence.

So, suppose that $i < j$, and let x be a shortest palindromic supersequence of $A[i..j]$. There are three possibilities.

Case 1. $A[i] \neq A[j]$. Since the first and last symbols of $A[i..j]$ are different, to create a shortest palindromic supersequence x of $A[i..j]$ we need to add either (a) the first symbol $A[i]$ at the end or (b) the last symbol $A[j]$ at the start. Furthermore, by a straightforward cut-and-paste argument, the remainder of x (excluding the added symbol and the symbol it matched) must be a shortest palindromic supersequence of the remainder of A (excluding the matched symbol); i.e., a shortest palindromic supersequence of $A[i+1..j]$ in case (a) where we added $A[i]$ to the end, or $A[i..j-1]$ in case (b) where we we added $A[j]$ to the start. Thus either $x = A[i]yA[i]$, where y is a shortest palindromic supersequence of $A[i+1..j]$; or $x = A[j]zA[j]$, where z is a shortest palindromic supersequence of $A[i..j-1]$ — whichever of the two is shorter. So, in this case, $P[i,j] = |x| = 2 + \min(|y|, |z|) = 2 + \min(P[i+1,j], P[i,j-1])$.

Case 2. $A[i] = A[j]$ and $i = j - 1$. In this case, $A[i..j]$ consists of two identical characters. So $A[i..j]$ is a palindrome of length 2, and it is its own shortest palindromic supersequence. Therefore, in this case, $P[i,j] = |A[i..j]| = 2$.

Case 3. $A[i] = A[j]$ and $i < j - 1$. Note that, in this case, $i + 1 \leq j - 1$ and so $A[i+1..j-1]$ is a nonempty sequence. So, $x = A[i]yA[j]$, where, y is a shortest palindromic supersequence of $A[i+1..j-1]$ (otherwise, we could find an even shorter supersequence of $A[i..j]$ than x that is a palindrome, contrary to the definition of x). Therefore $P[i,j] = |x| = 2 + |y| = 2 + P[i+1,j-1]$.

<u>*Solving the original problem:*</u> By the definition of the subproblems (∗), the required return value is simply $P[1..n]$.

<u>*Pseudocode:*</u> The algorithm is shown in pseudocode below. (The parts highlighted in grey are for part (b).)

```
    MinPalindromicSuperseq(A)
1   n := length(A)
2   for i := 1 to n do P[i,i] := 1 ; Q[i,i] = A[i]
3   for d := 1 to n − 1 do
4       for i := 1 to n − d do
5           j := i + d
6           if A[i] ≠ A[j] then
7               if P[i, j − 1] ≤ P[i + 1, j] then P[i,j] := P[i, j − 1] + 2 ; Q[i,j] := A[j] ∘ Q[i, j − 1] ∘ A[j]
8               else P[i,j] := P[i + 1, j] + 2 ; Q[i,j] := A[i] ∘ Q[i + 1, j] ∘ A[i]
            else
9               if i = j − 1 then P[i,j] := 2 ; Q[i,j] := A[i] ∘ A[j]
10              else P[i,j] := 2 + P[i + 1, j − 1] ; Q[i,j] := A[i] ∘ Q[i + 1, j − 1] ∘ A[j]
11  return ( P[1,n] , Q[1,n])
```

Running time: The running time of the algorithm is dominated by the doubly-nested loop in lines 3-6, which takes $O(n^2)$.

b. We also compute $Q[i,j]$, a shortest supersequence of $A[i..j]$ that is a palindrome, at the same time that we compute $P[i,j]$, the length of this supersequence. This is done in the part of the pseudocode highlighted in grey, where $\circ$ denotes the concatenation operation for sequences.