Homework 8 Solutions

December 5, 2019

6.4

You are given a string of n characters $s[1 \dots n]$, which you believe to be a corrupted text document in which all punctuation has vanished (so that it looks something like "itwasthebestoftimes ..."). You wish to reconstruct the document using a dictionary, which is available in the form of a Boolean function dict($\cdot$): for any string w,

$$dict(w) = \begin{cases} true & \text{if } w \text{ is a valid word} \\ false & \text{otherwise} \end{cases}$$

(a) Give a dynamic programming algorithm that determines whether the string $s[\cdot]$ can be reconstituted as a sequence of valid words. The running time should be at most $O(n^2)$, assuming calls to *dict* take unit time.

(b) In the event that the string is valid, make your algorithm output the corresponding sequence of words.

Solution

(a)

Subproblem: We define subproblem $S(i)$ for $0 \leq i \leq n$ where $S(i) = 1$ if the prefix $s[1...i]$ is a sequence of valid words; otherwise, $S(i) = 0$.

Initialization: It is sufficient to initialize $S(0) = 1$.

Recurrence equation: and update the values $S(i)$ in ascending order according to the recursion:

$$S(i) = \begin{cases} 1 & \text{if there exists } j < i \text{ such that } S(j) = 1 \text{ and } dict(s[j+1...i]) = true \\ 0 & \text{otherwise} \end{cases}$$

Then, the string s can be reconstructed as a sequence of valid words if and only if $S(n) = 1$.

Correctness: Consider $s[1 \dots i]$. If it is a sequence of valid words, there is a last word $s[j+1 \dots i]$, which is valid, and such that $S(j) = 1$ and the update will cause $S(i)$ to be set to 1. Otherwise, for any valid word $S[j+1 \ldots i]$, $S(j)$ must be 0 and $S(i)$ will also be set to 0.

Running Time: $O(n^2)$, as there are n subproblems, each of which takes time $O(n)$ to be updated with the solution obtained from smaller subproblems.

(b)

If $S(i)$ is updated to 1 by j^*, record $J[i] = j^*$. At termination, if $S(n) = 1$, backtrack the series of updates to recover the partition in words. This only adds $O(n)$ time to backtrack the array at the end. Hence, the running time remains $O(n^2)$.

6.7

A subsequence is palindromic if it is the same whether read left to right or right to left. For instance, the sequence

$$A, C, G, T, G, T, C, A, A, A, A, T, C, G$$

has many palindromic subsequences, including A, C, G, C, A and A, A, A, A (on the other hand, the subsequence A, C, T is *not* palindromic). Devise an algorithm that takes a sequence $x[1 \ldots n]$ and returns the (length of the) longest palindromic subsequence. Its running time should be $O(n^2)$.

Solution

Subproblem: We create an $n \times n$ matrix S (index starts from 1) and define $S[i, j]$ as the length of a longest palindromic subsequence within $x[i \ldots j]$ where $1 \leq i \leq j \leq n$.

Initialization: We set all elements of S to be 0.

Recurrence equation:

$$S[i, j] = \begin{cases} 1 & \text{if } i = j \\ 2 + S[i+1, j-1] & \text{if } x[i] = x[j] \text{ and } i < j \\ \max(S[i+1, j], S[i, j-1]) & \text{otherwise} \end{cases}$$

Start from $S[1, n]$ and do backtrack to find the longest palindromic subsequence.

Filling matrix S takes $O(n^2)$, do backtrack takes $O(n)$. Overall running time is $O(n^2)$.

Longest palindromic subsequence(x)

```
Input: a string x
Output: longest palindromic subsequence R
n = x.length
Create n × n matrix S, all elements of S are 0
for i = 1 to n
    S[i, i] = 1
// Dynamically programming to fill matrix
for a = 1 to n
    for j = a to n
        i = j − a + 1
        S[i, j] = { 1                              if i == j
                  { 2 + S[i + 1, j − 1]            if x[i] = x[j] and i ≠ j
                  { max(S[i + 1, j], S[i, j − 1])  otherwise
Create string R and R.length = S[1, n]
// Backtracking
i = 1, j = n, t = 1
while i ≠ j and i ≥ 1, j ≤ n
    if x[i] == x[j]
        // Found same characters
        R[t] = x[i], R[n − t + 1] = x[i]
        t = t + 1
        i = i + 1, j = j − 1
    else if x[i] ≠ x[j] and S[i, j] == S[i + 1, j]
        if i + 1 == j
            R[t] = x[i]
        i = i + 1
    else
        if j − 1 == i
            R[t] = x[j]
        j = j − 1
Return R
```

6.22

Give an $O(nt)$ algorithm for the following task.

Input: A list of n positive integers $a_1, a_2, \ldots, a_n$; a positive integer t.

Question: Does some subset of the a_i's add up to t? (You can use each a_i at most once.)

(Hint: Look at subproblems of the form "does a subset of $\{a_1, a_2, \ldots, a_i\}$ add up to s?")

Solution

Subproblem: Does some subset of a_1 to a_i $(i \leq n)$ add up to s $(\leq t)$? We record the answer in matrix X entry $X[i,s]$. $X[i,s]$=True if the answer is yes; otherwise $X[i,s]$=False.

Initialization:

$$\begin{cases} X[0,0] = \text{True} & \\ X[0,s] = \text{False} & s > 0 \\ X[i,s] = \text{False} & s < 0 \end{cases}$$

Recurrence equation:

$$X[i,s] = X[i-1, s-a_i] \text{ or } X[i-1,s]$$

$X[n,t]$ contains the answer.

Running time: Filling in each entry in the matrix takes constant time. So the runtime is the big-Oh of the matrix size, i.e. $O(nt)$.

Pseudocode

```
SubsetAddup(a_1, a_2, ... a_n, t)
Input: n positive integers a_1, a_2, ... a_n, a positive integer t
Output: True or False

Create an (n + 1) × (t + 1) matrix X, all elements of X are False
X[0, 0] = True

for s = 0 to t
    for i = 1 to n
        if X[i − 1, s] == True
            X[i, s] = True
        else if s − a_i < 0
            X[i, s] = False
        else
            X[i, s] = X[i − 1, s − a_i]

return X[n, t]
```