Illinois Institute of Technology
Department of Computer Science

Revised Solutions to Homework Assignment 0

CS 535 Design and Analysis of Algorithms
Fall Semester, 2012

1. We want to break a string into pieces using a basic operation STRING-SPLIT(s, k, f, b) that splits a string s into two pieces: the first k characters f, and b the remaining characters; this basic operation requires copying the string, so it costs $O(|s|)$ time to break string s characters into two pieces. What we really want to do, however, is to break a string into many pieces using successive applications of STRING-SPLIT. The order in which the breaks are made affects the total amount of time usedfor example, suppose we need to break a 20-character string after characters 3, 8, and 10 (numbering the characters in ascending order from the left, starting from 1). If the breaks are made in left-to-right order, then the first break costs 20 units of time, the second break costs 17 units of time, and the third break costs 12 units of time, a total of 49 units of time. If the breaks are made in the right-to-left order, then the first break costs 20 units of time, the second break costs 10 units of time, and the third break costs 8 units of time, a total of 38 units of time.

   (a) Devise and analyze an un-memoized dynamic programming algorithm that, when given a the numbers of characters after which to break, determines the cheapest cost of those breaks.

   (b) Show how to memoize your algorithm in (a) and analyze the resulting memoized algorithm. The result should take polynomial time.

   Solution:

   Let S be a string of length n and let $S(i,j)$ denote the cheapest cost of breaking the substring $S[i...j]$ given string S. Let $L[1...m]$ denote the array containing m break points. The idea is as follows: given a string, we find an index $L[k]$ such that breaking the string at $L[k]$ is the cheapest-cost way. If $L[k]$ is found then the break costs $j-i$ units of time. If there is no such k in L array which can break current string, we return 0.

$$S(i,j) = \begin{cases} min\{S(i, L[k]) + S(L[k]+1, j) + j - i + 1\} & \text{if } \exists k : i \leq L[k] < j \\ 0 & \text{otherwise} \end{cases}$$

   (a) We first design an un-memoized algorithm as follows:

Algorithm 1 The subroutine $S(i,j)$ for un-memoized method

```
if ∄k : i ≤ L[k] < j then
  Return 0
else
  for each k : i ≤ L[k] < j do
    S(i, j) = min{S(i, L[k]) + S(L[k] + 1, j) + j − i + 1}
  end for
  Return S(i, j)
end if
```

So we just need to call the subroutine $S(1,n)$ to get the result. In this algorithm, we actually calculate each permutation of the breaking points in $L[1...m]$, so the time complexity is $O(m!)$.

(b) We next design a memoized algorithm that improves the time complexity. For each $S(i, j)$ in the memoized algorithm, we only calculate once. So we need a $n \times n$ matrix to restore intermediate results. The matrix is denoted as $M_{n\times n}$, that is $M(i, j)$ stores the value of $S(i, j)$.

Algorithm 2 The subroutine $S(i, j)$ for memoized method

```
if S(i,j) ≠ NULL then
  Return S(i,j)
else
  if ∄k : i ≤ L[k] < j then
    Return 0
  else
    for each k : i ≤ L[k] < j do
      S(i,j) = min{S(i,L[k]) + S(L[k]+1,j) + j − i + 1}
    end for
    Return S(i,j)
  end if
end if
```

So we can just call $S(1, n)$ to get the result. Note that in this algorithm, we only need to calculate each $S(i, j)$ once, and in the worst case the time complexity for calculating one cell would be $O(m)$, so the time complexity will be $O(mn^2)$ in the worst case. We can also design a bottom-up version of the memoized algorithm as follows.

Algorithm 3 The bottom-up memoized method

```
for i = 1 to n do
  S(i,i) = 0
end for
for l = 2 to n do
  for i = 1 to n − l + 1 do
    j = i + l − 1
    S(i,j) = ∞
    if ∄k : i ≤ L[k] < j then
      S(i,j) = 0
    else
      for each k : i ≤ L[k] < j do
        S(i,j) = min{S(i,L[k]) + S(L[k],j) + j − i + 1}
      end for
    end if
  end for
end for
Return S(1,n)
```

2. Find counter examples to the following heuristics for the string-cutting problem above. That is, find a string and places to cut such that when cuts are made in the order given, the cost is higher than the optimal.

   (a) Start by cutting the string as close to the middle as possible, and the repeat the same thing on the resulting pieces.

(b) Start by making (at most) two cuts to separate the smallest substring. Repeat this until finished.

(c) Start by making (at most) two cuts to separate the largest substring. Repeat this until finished.

Solution:

(a) Suppose we need to cut a 20-character string after characters 5, 10 and 11. The cheapest cost is $20 + 11 + 6 = 37$ by cutting the string in the order 11, 5, 10. When cuts are made in the order 10, 5, 11, the cost is $20 + 10 + 10 = 40$ which is higher than the optimal.

(b) Suppose we need to cut a 20-character string after characters 5, 10 and 11. The cheapest cost is $20 + 11 + 6 = 37$ by cutting the string in the order 11, 5, 10. When cuts are made in the order 10, 11, 5 the cost is $20 + 10 + 10 = 40$ which is higher than the optimal.

(c) Suppose we need to cut a 20-character string after characters 5, 11 and 13. The cheapest cost is $20 + 11 + 9 = 40$ by cutting the string in the order 11, 5, 13. When cuts are made in the order 13, 5, 11, or 13, 11, 5 the cost is $20 + 13 + 8 = 41$ or $20 + 13 + 11 = 44$ which are higher than the optimal.