CSCI-UA.0310-004/005 Basic Algorithms April 8, 2015

Solutions to Problem 1 of Homework 8 (6 points)

Name: Jason Yao *Due: Wednesday, April 8*

Using dynamic programming, find the optimum printing of the text "I am a big fan of rats", i.e. $\ell_1 = 1, \ell_2 = 2, \ell_3 = 1, \ell_4 = 3, \ell_5 = 3, \ell_6 = 2, \ell_7 = 4$, with line length $L = 10$ and penalty function $P(x) = x^3$. Make sure you justify all your steps (and not just state the answer without proof). Will the optimal printing you get be consistent with the strategy "print the word on as long as it fits, and otherwise start a new line"?

Solution:

$P(x) = x^3$, $L = 10$
$m(i)$ = minimized penalty
$m(0) = 0$
$m(1)$ = Printed "I", penalty = p(L - ℓ_1), penalty = $9^3 = 729$
$m(2)$ = Printed "I am", penalty = p(L - (ℓ_1 + ℓ_2), penalty = $6^3 = 216$
$m(3)$ = Printed "I am a", penalty = p(L - (ℓ_1 + ℓ_2 + ℓ_3), penalty = $4^3 = 64$
$m(4)$ = Printed "I am a big", penalty = p(L - (ℓ_1 + ℓ_2 + ℓ_3 + ℓ_4), penalty = $0^3 = 0$
$m(5)$ = min(m[4] + "fan", m[3] + "big fan", m[2] + "a big fan")
= min($0 + 7^3$, $64 + 3^3$, $216 + 1^3$)
= min(343, 73, 217)
= "I am a" / "big fan", penalty of 73
$m(6)$ = min(m[5] + "of", m[4] + "fan of", m[3] + "big fan of")
= min($4^3 + 3^3 + 8^3$, $0 + 6^3 + 0$, $4^3 + 0 + 0$)
= min($64 + 27 + 512$, $0 + 64 + 0$, $64 + 0 + 0$)
= min(603, 64, 64)
= "I am a" / "big fan of" or "I am a big / fan of" , penalty is 64
$m(7)$ = min(m[6] + "rats", m[5] + "of rats")
= min($64 + 6^3$, $64 + 3^3 + 3^3$)
= min(280, 118)
= "I am a / big fan/ of rats", with penalty 118

This is the optimal solution (minimizing the penalty), since the greedy solution would have been "I am a big / fan of / rats", with a penalty of (0 + 64 + 216) = 280, which is much higher than the solution that we found.

□

CSCI-UA.0310-004/005 Basic Algorithms April 8, 2015

Solutions to Problem 2 of Homework 8 (10 points)

Name: Jason Yao *Due: Wednesday, April 8*

You have $m \times n$ chocolate bar. You are also given a matrix $\{p[i,j] \mid 1 \le i \le m, 1 \le j \le n\}$ telling you the price of the $i \times j$ chocolate bar. You are allowed to repeat the following procedure any number of times, starting initially with the single big $m \times n$ piece you have. Take one of the pieces you have and split it into two pieces by cutting it either vertically or horizontally. Say, $m = 5$, $n = 4$. You may first choose to split it into two pieces of size 3×4 and 2×4. Then you may take the 3×4 piece and split it into two pieces 3×3 and 3×1. Finally, you may take the previous 2×4 piece and split it into two 1×4 pieces. If you stop, you have four pieces of sizes 3×3, 3×1, 1×4 and 1×4, which you can sell for $p[3,3] + p[3,1] + 2p[1,4]$. You goal is to find a partition maximizing your total profit.

(a) (5 points) Let $C[i,j]$ be the largest profit you can get by splitting an $i \times j$ piece, where $0 \le i \le m$, $0 \le j \le n$ and we set $C[i,0] = C[0,j] = 0$. Write a recursive formula for $C[m,n]$ in terms of values $C[i,j]$, where either $i < m$ or $j < n$.

Solution:

```
c[m + 1][n + 1]

maxProfitDriver()
  maxProfit(0,0)
  return c[m][n]
end maxProfitDriver

void maxProfit(i, j)
  if ((i == 0) || (j == 0))
    c[i][j] = 0
  for x = 1 to j - 1
    for y = 1 to i - 1
      tempVert = c[i][x] + c[i][j - x]
      tempHor = c[y][j] + c[i - y][j]
      if (tempVert > maxVertical)
        maxVertical = tempVert
      if (tempHor > maxHorizontal)
        maxHorizontal = tempHor
    endfor
  endfor

  actualMax = max(maxVertical, maxHorizontal)
  if (actualMax > p[i][j])
    c[i][j] = actualMax
```

```
  else
    c[i][j] = p(i)(j)
endmaxProfit
```

□

(b) (5 points) Write a bottom-up procedure to compute $C[m, n]$ and analyze its running time as a function of m and n.

Solution:

```
c[m + 1][n + 1]

maxProfitDriver()
  maxProfit(0,0)
  return c[m][n]
end maxProfitDriver

void maxProfit(i, j)
  if ((i == 0) || (j == 0))
    c[i][j] = 0
  for x = 1 to j - 1
    for y = 1 to i - 1
      tempVert = c[i][x] + c[i][j - x]
      tempHor = c[y][j] + c[i - y][j]
      if (tempVert > maxVertical)
        maxVertical = tempVert
      if (tempHor > maxHorizontal)
        maxHorizontal = tempHor
    endfor
  endfor

  actualMax = max(maxVertical, maxHorizontal)
  if (actualMax > p[i][j])
    c[i][j] = actualMax
  else
    c[i][j] = p(i)(j)
endmaxProfit
```

□

CSCI-UA.0310-004/005 Basic Algorithms April 8, 2015

Solutions to Problem 3 of Homework 8 (8 points)

Name: Jason Yao *Due: Wednesday, April 8*

Consider the following greedy solution for the above "Dividing Chocolate" problem. Given some piece of size $i \times j$, and some proposed (either horizontal or vertical) cut of this piece, we say that the cut is "locally improving" if the sum of the prices for two resulting sub-pieces is bigger then the price $p[i, j]$ of the original uncut piece. The *best locally improving* (i, j)*-cut* is then the cut maximizing the difference between the sum of the prices of the two pieces and the original price $p[i, j]$ (if there are ties, any of the choices is fine).

The algorithm now proceeds as follows. Starting from the original $m \times n$ piece, it finds the best locally improving (m, n)-cut, and then recursively finds the best locally improving cuts for the resulting pieces, until no such locally improving cuts are possible (i.e., all the remaining pieces cannot be subdivided further in a locally improving way).

Show that this greedy solution is not a good solution by finding a counter-example. Make sure your counter-example is non-trivial: namely, $p[i, j] \geq p[k, t]$ where $i \geq k$ and $j \geq t$ (i.e., if one piece is included in a bigger piece, its price must be smaller).

Solution:

Proof by counter example:

Let m = n = 2

p(1,1) = 2

p(1,2) = p(2,1) = 3

p(2,2) = 6

optimalPrice = cut at p(1,1) = 8.

If we utilized the greedy algorithm, it would end up utilizing p(2,2) due to locally improving, compared to the optimal solution.

□

CSCI-UA.0310-004/005 Basic Algorithms April 8, 2015

Solutions to Problem 4 of Homework 8 (10 points)

Name: Jason Yao *Due: Wednesday, April 8*

Recall the dynamic programming solution for the matrix chain multiplication problem. Here we give two greedy candidate solutions for this problem. For each of the proposed solutions find a counter-example proving that it is not correct.

(a) (5 points) In each step we will "get rid" of the biggest possible dimension p_i (intuitively, we want to "pay" for such huge p_i only once). Formally, let p_i be the biggest value of dimension: i.e., the matrix A_i has dimension $p_{i-1} \times p_i$ and the matrix A_{i+1} has dimension $p_i \times p_{i+1}$, where $p_j \leq p_i$ for all j. Then we will multiply A_i with A_{i+1} first. After that we repeat the same greedy procedure on the remaining $n-1$ matrices. And so on until we get the final result.

Solution:

Let there be matrices A_1, A_2, A_3 such that

A_1 is a (1x3) matrix

A_2 is a (3x4) matrix

A_3 is a (4x3) matrix

□

(b) (5 points) In each step choose such a pair of adjacent matrices A_i and A_{i+1} such that the cost of multiplication them is the smallest possible at this point. Namely, $p_{i-1}p_ip_{i+1} \leq p_{j-1}p_jp_{j+1}$, for all j. Then multiply A_i and A_{i+1} first. After that we repeat the same greedy procedure on the remaining $n-1$ matrices. And so on until we get the final result.

Solution:

Let there be matrices A_1, A_2, A_3 such that

A_1 is a (4x1) matrix

A_2 is a (1x7) matrix

A_3 is a (7x2) matrix

□