Homework 6
Problem solutions
cs161
Summer 2009

Problem 1 (8 points):

a).There was a bit of confusion about this problem, we are going to use the definition of Levenshtein distance that does not allow multiple swaps for one character (i.e. abc → acb → cab not allowed). So we only do the swap if the last two characters in the string are "switched" from each other.

Hence for swap, the recursion is: EDIT(X,Y) = 1+EDIT($X_{1..m-2}$,$Y_{1,n-2}$) if $X_{m-1} = Y_n$ and $X_m = Y_{n-1}$

b). Just add this recursion to the list of things to minimize over, but make sure that you only apply it when $X_{m-1} = Y_n$ and $X_m = Y_{n-1}$

If you solved it the original way, that will be correct as well.

Problem 2 (10 points):

a).

Naive:
Naïve is best if the first character or (characters) of the patterns do not exist in the string
The naive method would outperform the other two. The naive method would run at O(n-m) time. While the other two will at best require linear time for the searching and will require preprocessing time.
An application where this might occur is a large searchable data base of science text books, where most searches involve obscure scientific terms

FSA:
FSA is best if we know we're likely to find lots of repeated, overlapping patterns in a long string, and we want to count the number of times those patterns appear. This is because it takes linear time to traverse through the string, and the FSA gives us a natural way to keep track of patterns and overlaps.
An application in which this situation might occur is if you're doing a frequency analysis of a binary encoded file, so that you can compress the file by using a few bits to represent larger patterns. It's likely in the binary file that lots of repeated patterns like "0110" will appear.

Rabin-Karp:
Rabin-Karp is used for multiple pattern matching. It can handle multiple large search terms by using a hashing function. This makes it ideal in detecting plagiarism.

b). Describe another algorithm and describe the best case scenario:
Boyer-Moore, check out this link: http://www.cs.utexas.edu/~moore/best-ideas/string-searching/

Problem 3 (10 points):

Our string data structure will have the following 3 components:
a). The length of the string stored in an integer
b). A character array that contains the string
c). A hash table that maps each letter that occurs in the string to a linked list of integer indices of positions in the string where the letter occurs

For example, the string "RECEIVE" would be stored as follows:
len = 7
Str = [R,E,C,E,I,V,E]
H = R → 1
E → 2 -- > 4 --> 7
C → 3
I → 5
V → 6

Length: Just return len, obviously O(1) operation

Concatenation: To concatenate two strings s1 and s2 stored using our data structure. Set the new len to be the sum of the lengths of a and b. Create an array with length equal to the new len and copy the arrays from s1 and s2 next to each other. Also, merge the two hash tables together by appending linked lists if there are common keys. Overall this will take O(n+m) operations assuming constant time hashing.

Substitution: If we want to substitute character b for character a, find the linked list for a in H (This takes constant time), if there is already a linked list for b, append the first linked list to the linked list for b, otherwise, replace the key "a" with "b". Now, replace the corresponding entries in the Str array using indices from the linked list for b. This results in an O(j) solution.

Alternatively, we can not store b), the original array. This allows us to do substitution in O(1) time, but does have consequences for other operations not listed. This answer is also acceptable since it satisfies the problem constraints.

Problem 4 (10 points):

Define P_{KC} as the amount of pancakes (in tons) to ship from Kansas to California. Similarly, define P_{MC} as the amount to ship from Mexico to California. P_{KN} and P_{MN} are also defined similarly.

Hence we have the following linear program:

$$\text{Minimize } 4\,P_{MN} + P_{MC} + 2\,P_{KN} + 3\,P_{KC}$$

With the following constraints:

$$\begin{aligned} &P_{MC} + P_{MN} = 8 \\ &P_{KC} + P_{KN} = 15 \\ &P_{KN} + P_{MN} = 10 \\ &P_{KC} + P_{MC} = 13 \\ &P_{KC},\ P_{MC},\ P_{KN},\ P_{MN} \geq 0 \end{aligned}$$

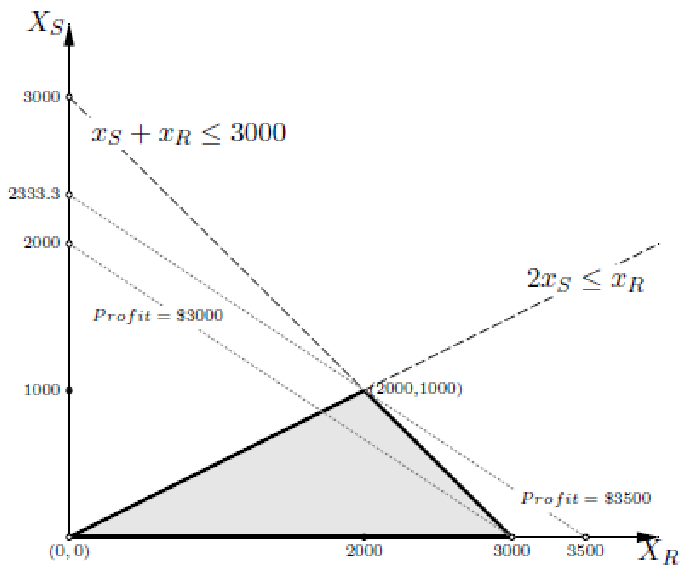
Extra Credit [5 points]

Lets denote the amount of Regular Duff Beer Moe orders by x_R and the amount of Strong Duff Beer Moe orders by x_S.
Profit earned by Moe is given as follows:

$Profit = x_R + 1.5x_S$, which needs to be maximized subject to the following constraints:

$2x_S \leq x_R$	twice as much regular beer as strong beer
$x_S + x_R \leq 3000$	3000 pints in total
$x_S, x_R \geq 0$	positive amounts of beer can be bought

Solving it geometrically gives a vertex at (2000, 1000) which will maximize the objective. Therefore, we should get 2000 pints of Regular Duff and 1000 pints of Duff Strong which will give us \$3500 in profit.


X_S
$x_S + x_R \leq 3000$
$2x_S \leq x_R$
$Profit = \$3000$
(2000,1000)
$Profit = \$3500$
(0, 0)
X_R