CMPS 102: Homework #6

Due on Tuesday, May 19th, 2015

John Allard 1437547

Problem 1

Determine the longest common subsequence of $x = (1, 0, 0, 1, 0, 1, 0, 1)$ and $y = (0, 1, 0, 1, 1, 0, 1, 1, 0)$ Build the table $C(i, j)$ of the dynamic programming algorithm for these two strings, where $C(i, j)$ denotes the length of the longest common subsequence between $x_1x_2...x_i$ and $y_1y_2...y_j$. Also produce the table of "arrows" that lets you recover the solution.

		1	0	0	1	0	1	0	1
	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1
1	0	1	1	1	2	2	2	2	2
0	0	1	2	2	2	3	3	3	3
1	0	1	2	2	3	3	4	4	4
1	0	1	2	2	3	3	4	4	5
0	0	1	2	3	3	4	4	5	6
1	0	1	2	3	4	4	5	5	6
1	0	1	2	3	4	4	5	5	6
0	0	1	2	3	4	5	5	6	6

		1	0	0	1	0	1	0	1
	0	0	0	0	0	0	0	0	0
0	**0**	l	d	d	l	d	l	d	l
1	0	**d**	**l**	l	d	l	d	l	d
0	0	u	d	**d**	l	d	l	d	l
1	0	d	u	l	**d**	**l**	d	l	d
1	0	d	u	u	d	u	**d**	l	d
0	0	u	d	d	u	d	u	**d**	l
1	0	d	u	u	d	u	d	**u**	d
1	0	d	u	u	d	u	d	u	**d**
0	0	u	d	d	u	d	u	d	**u**

Length of longest common sub-sequence : 6
Longest common subsequence : 101101 (one of many, also 100110 and 100111)

Problem 2

Suppose we are given three strings of characters $X = x_1x_2...x_m$, $Y = y_1y_2...y_n$, and $Z = z_1z_2...z_m + n$. Z is said to be a "shuffle" of X and Y if Z can be formed by interspersing the characters from $X$and Y in a way that maintains to left-to-right ordering of the characters from each string. For example the, cchocohilaptes is a shuffle of chocolate and chips, but chocochilatspe is not. Devise a dynamic programming algorithm that takes as input X,Y,Z,m, and n, and determines whether or not Z is a shuffle of X and Y. Analyze the worst-case running time and space requirements of your algorithm.

Hint: The values in your table should be Boolean, not numeric.
Answer :
The algorithm to solve this problem will depend on 2 variables, the indices of the characters in X and Y. We can calculate the index of the character we're comparing for the current iteration in Z by adding the indices of the current characters we're looking at it and X and Y and subtracting one.
The subproblems are as follows : If the current problem is on the strings $X[1 : i]$, $Y[1 : j]$, $Z[1 : k]k = i+j-1$,

the subproblems consist of seeing if the strings $Z[1:k-1], X[1:i-1], Y[1:j]$ or $Z[1:k-1], X[1:i], Y[1:j-1]$ are shuffles of one another. Given the answer to these subproblems, we can find the solution to the current-length strings by seeing if the character Z_k can be made by taking either X_i or Y_j and combining it with the answers to the 2 subproblems. This will be made specific below.
The table in this case is boolean (as was suggested in the hint), with dimensions $m \times n$ The bool in table entry (i, j) represents if the either of the characters X_i or Y_j can be used to make the character $Z_k : k = i + j - 1$. The arrows for this table either go right or left, depending on which string has the character that matches the character in Z. If Z_k is matches by a character on the x axis, we travel left, if it is matches by a character on the y axis we go up, and if they both match we can go either way.

The formal recurrance relation is as follows : Given strings X of length m, Y of length n, and Z of length $n+m$, $S(p,q) : (m,n) \geq (p,q) \geq (0,0)$ takes the indices of two characters in X and Y and returns a boolean value. S is defined as :

$$S(p,q) = [(Z_k === X_p \text{ AND } S(p-1,q)) \text{ OR } (Z_k == Y_q \text{ AND } S(p,q-1))]$$

We also have the base cases that if p or q becomes 0, we simply check other non-zero index. If they're both zero we return true.

This says that at index $k = p+q$, we either match the character X_p and recurse on Z with one less character, X with one less character, and Y as it is, or we do the same except with Y.
The algorithm doesn't actually need to build the algoritmh

```
# X, Y, Z = strings of len (m, n, m+n-1)
# @args - p, q are the X, Y indices we are currently querying.
checkShuffle(M, X, Y, Z) :
  p = 1
  q = 1
  # note the short-circuit AND evalutation to avoid indexing out of X or Y
  while (p+q) < (m+n) :
    k = p+q-1
    if p <= m AND Z[k] == X[p] :
      p = p+1
    else if q <= n AND Z[k] == Y[q] :
      q = q+1
    else :
      return false

  return true
```

The running time for this algorithm is linear in m and n, which means it is linear in the size of Z. If this was the strictly harder version of this problem where a matching character in Z and count for both a match in X and Y is a strictly harder problem and wouldn't be able to be solved so easily. Notice the algorithm doesn't even need to fill in the table, it can find a path through it without needing to fill in the entire thing. This is because once it finds the character X_p cannot match at position Z_k, it knows that no character $X_{p+n} : n > 0$ can be placed in the spot either without violating the ordering of X. The algorithm runs a single for-loop from iteration number 1 to iteration k, increasing by one each time. If it runs into a wall it returns false right away.

Problem 3

Suppose that you are given an n x n checkerboard and a checker. You must move the checker from the bottom edge of the board to the top edge of the board according to the following rule. At each step you may move the checker to one of three squares:

1. the square immediately above,
2. the square that is one up and one to the left (but only if the checker is not already in the leftmost column).
3. the square that is one up and one to the right (but only if the checker is not already the rightmost column).

Each time you move form square x to square y, you receivep(x,y) dollars. Your are given p(x,y) for all pairs (x,y) for which a move from x to y is legal. Do not assume that p(x,y) is positive.
Give an algorithm that figures out the set of moves that will move the checker from somewhere along the bottom edge to somewhere along the top edge while gathering as many dollars as possible. Your algorithm is free to pick any square along the bottom edge as a starting point and any square along the top edge as a destination in order to maximize the number of dollars gathering along the way. What is the running time of your algorithm?

Problem 4

Extra Credit: Viterbi algorithm.We can use dynamic programming on a directed graph $G = (V, E)$for speech recognition. Each edge (u, v) in E is labeledwith a sound $s(u, v)$ from a finite set S of sounds. Thelabeled graph is a formal model of a person speaking arestricted language. Each path in the graph starting from adistinguished vertex v_0 in V corresponds to a possible sequence of sounds produced by the model. The label of a directed path is defined to be the concatenation of the labels of the edges on that path.
a) Describe an efficient algorithm that, given an edge-labeled graph G with distinguished vertex v_0 and a sequence $L = (s_1, s_2, ..., s_k)$ of characters from S, returns a path in G that begins at v_0 and has L as its label, if any such path exists. Otherwise, the algorithm should return NO-SUCH PATH. Analyze the running time of your algorithm.

Now, suppose that every edge (u, v) in E has also been given an associated nonnegative probability p(u, v) of traversing the edge (u, v) from vertex u and thus producing the corresponding sound. The sum of the probabilities of the edges leaving any vertex equals 1. The probability of a path is defined to be the product of the probabilities of its edges. We can view the probability of a path beginning at v_0 as the probability that a "random walk" beginning at v_0 will follow the specified path, where the choice of which edge to take at a vertex u is made probabilistically according to the probabilities of the available edges leaving u.

b) Extend your answer to part a) so that if a path is returned, it is a most probable path staring at v_0 and having label L. Analyse the running time of your algorithm.