CMPS 102: Homework #7

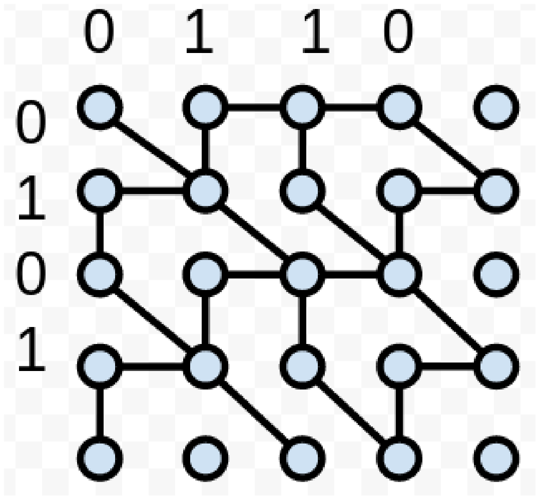
Due on Tuesday, May 26th, 2015

John Allard 1437547

Problem 1

1) Consider the longest common subsequence problem between two strings $x_1, ..., x_n$ and $y_1, ..., y_m$. Define a graph over the nxm grid (plus possibly some vertices around the edges) s.t. the longest common subsequence corresponds to the longest path in this graph. - Clearly describe the condition for the presence of an edge between two vertices on the grid.
- How should the edges be labeled?
- How do you find the longest path?
- Is this algorithm more efficient than the dynamic programming algorithm?

Answer : We will solve this problem using a directed graph over $m \times n$ vertices that are labeled according to their coordinates $(p, q) : 0 \leq p \leq m$ and $0 \leq q \leq n$. Edges are generated as follows : if letters p and q are the same, draw an edge from $V(p, q)$ to $V(p+1, q+1)$, aka a diagonal. If the letters are different, draw two edges from $V(p, q)$, one to $V(p+1, q)$ and the other to $V(p, q+1)$, aka down and right. The import part is this : edges that go diagonal are given a weight of 1, and edges that go right or down get a weight of 0. The longest path now doens't depend on the number of edges directly, but rather how many diagonal edges are along a path, since they contribute all of the weight.



Graph of `0101` and `0110`.

Because all edges travel in either the positive x, positive y, or positive x and y directions, we know that this graph is acyclic. Because this graph is directed and acyclic, we can use a realtively fast algorithm (compared to the NP-hard algorithm for the longest-path on a general-graph case) to find the longest path. The algorithm contains two parts, making the graph and retreiving the longest path. To find the longest path, we have to find a topological sort of the graph.
This can be done with a modifed version of DFS that pushes vertices onto a stack, at the very end that stack will contain the topological sort of G. Once we have this sort, we can find the longest path using a simple algorithm.

Start by walking over all $m \times n$ vertices in the graph and generating edges. Each vertex can have at most two edges eminated from it, so each of the $m \times n$ operation are constant. Thus constructing the graph is $O(mn)$. Once the graph is constructed, DFS is run on it which runs in time $O(|V + |)$, since $E \leq 2 * (V)$, DFS runs in time $O(V) = O(mn)$. The algorithm in short walks along the topolical sort from the source vertex (which would be $V(0, 0)$ for the shortest-common-subsequence problem), and updates its neighbors whereever it can by adding its own weight-sum to the weight of the incident edges and setting the neighbors weight-sum to this value.

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

Problem 2 continued on next page...

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

Problem 2 continued on next page. . .

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.