COMS 311: Homework 5
Due: April 26^{th}, 11:59pm
Total Points: 30

Late submission policy. Any assignment submission that is late by not more than two business days from the deadline will be accepted with 20% penalty for each business day. That is, if a homework is due on Friday at 11:59 PM, then a Monday submission gets 20% penalty and a Tuesday submission gets another 20% penalty. After Tuesday no late submissions are accepted.

Submission format. Homework solutions will have to be typed. You can use word, LaTeX, or any other type-setting tool to type your solution. Your submission file should be in pdf format. Do **NOT** submit a photocopy of handwritten homework except for diagrams that can be hand-drawn and scanned. We reserve the right **NOT** to grade homework that does not follow the formatting requirements. Name your submission file: `<Your-net-id>-311-hw5.pdf`. For instance, if your netid is `asterix`, then your submission file will be named `asterix-311-hw5.pdf`. Each student must hand in their own assignment. If you discussed the homework or solutions with others, a list of collaborators must be included with each submission. Each of the collaborators has to write the solutions in their own words (copies are not allowed).

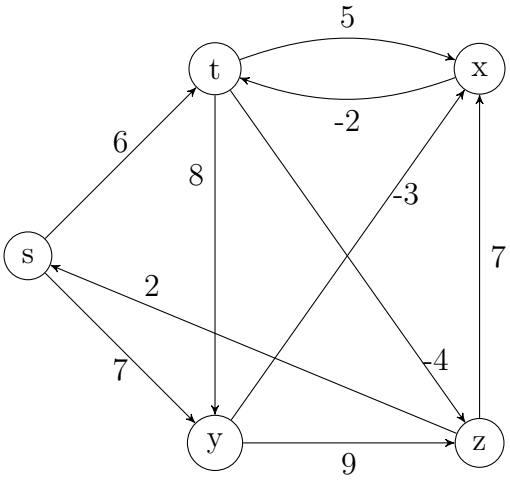
General Requirements

- When proofs are required, do your best to make them both clear and rigorous. Even when proofs are not required, you should justify your answers and explain your work.
- When asked to present a construction, you should show the correctness of the construction.

Some Useful (in)equalities

- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $2^{\log_2 n} = n$, $a^{\log_b n} = n^{\log_b a}$, $n^{n/2} \leq n! \leq n^n$, $\log x^a = a \log x$
- $\log(a \times b) = \log a + \log b$, $\log(a/b) = \log a - \log b$
- $a + ar + ar^2 + ... + ar^{n-1} = \frac{a(r^n-1)}{r-1}$
- $1 + \frac{1}{2} + \frac{1}{2^2} + ... + \frac{1}{2^n} = 2(1 - \frac{1}{2^{n+1}})$
- $1 + 2 + 4 + ... + 2^n = 2^{n+1} - 1$

1. (**10 pts**) Consider the following directed graph.



Apply Bellman-Ford algorithm to find the shortest distances to all vertices from the source s. Your solution must present the dictionary entries for each vertex for each iteration (as discussed in class lecture).

iteration	s	t	x	y	z
setup	0	inf	inf	inf	inf
1	0	6	inf	7	inf
2	0	6	4	7	2
3	0	2	4	7	2
4	0	2	4	7	-2

2. Given two strings, write an algorithm for finding the longest common subsequence. A subsequence of a string is a sequence of characters which conforms to the relative ordering of the characters in the string. The characters do not necessarily appear contiguously in the string.

   Example: `GHTCCHT` and `CHTGCH`. The longest common subsequence is `HTCH`.

   Design an algorithm using a Dynamic Programming strategy which consists of the following:

   (a) (**7 pts**) Formalize a recursive definition for the solution.

   The problem can be broken up into sub problems comparing one character at a time. Given Strings A, B, if A[i] == B[j] for some i, j, then A[i] will be part of the subsequence. i and j will be used to iterate through the string.

   This recursive definition would look something like (code snippet):

```
int i, j = 0
while (i < len(A), j < len(B)):
    if(A[i] == B[j]):
        return A[i] + LCS(A, B, i, j)
    else:
        return max {LCS(A, B, i + 1, j), LCS(A, B, i, j + 1)}
```

   In this recursive definition, we are checking to see if the characters are the same at indexes i, j. If they are equal, they are part of the subsequence. If they are not, we must increment either i or j, so we will keep the better outcome of the two.

(b) (**10 pts**) Write pseudocode for an iterative dynamic programming algorithm with a dictionary to implement the solution.

```
LCS(String A, String B, int m, int n):
    if (len(A) == 0 OR len(B) == 0):
        return 0 // null input check

    int[][] dict = new int[m + 1][n + 1]
    for (i = 0 : m + 1):
        for (j = 0 : n + 1):
            if(A[i] == A[j]):
                dict[i][j] = dict[i + 1][j + 1] + 1
            else:
                dict[i][j] = max{ dict[i + 1][j], dict[i][j + 1]}

    return dict
```

(c) (**3 pts**) Analyze the runtime of your algorithm.

This algorithm uses nested for loops of size m and n, meaning this runtime will be O($m \cdot n$), where m and n are $|A|$ and $|B|$ respectively, so this algorithm will finish in O($|A| \cdot |B|$) time, where A and B are the input strings.