# 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).

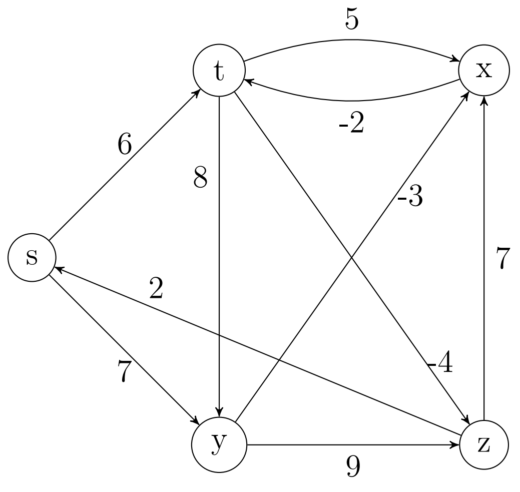
## General Requirements

- 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 |
|---|---|---|---|---|---|
| 0 | 0 | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 1 | 0 | 6 | $\infty$ | 7 | $\infty$ |
| 2 | 0 | 6 | 4 | 7 | 2 |
| 3 | 0 | 2 | 4 | 7 | 2 |
| 4 | 0 | 2 | 4 | 7 | -2 |
| 5 | 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.

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

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

First, consider the optimization objective: find the length of the longest common subsequence. Second, formalize a recursive definition for the solution. Third, realize a dynamic programming algorithm to implement the solution (iterative algorithm with a dictionary). Finally, compute the longest common subsequence from the dictionary.

**Objective:** find the length of the longest common subsequence.

Consider a function which takes as input the two strings $s$ and $t$ containing $m$ and $n$ characters, respectively and outputs the length of the longest common subsequence. We will represent the function as `LCS(m, n)`. Therefore,

$$LCS(m,n) = \begin{cases} 0 & \text{if } m = 0 \vee n = 0 \\ 1 + LCS(m-1, n-1) & \text{if } s[m] == t[n] \\ \texttt{max}\{LCS(m-1,n), LCS(m,n-1)\} & \text{otherwise} \end{cases}$$

The first case corresponds to the base case, when either one of the string is an empty string. The second case corresponds to the case when the last characters of the strings are equal. In the final case, the last characters of the strings are not the same. In this scenario, there are two possibilities to consider: (a) the last character of first string may match with the second last character of the second string or (b) the last character of the second string may match with the second last character of the first string.

The above recursive formulation shows the dependencies between the subproblems; the $LCS$ for smaller values for $m$ and $n$ should be computed before the larger values. This results in the following iterative algorithm using DP strategy.

```
// dict[i][j]: holds the LCS(i, j) valuations
for i = 0 to m
    for j = 0 to n
        if i == 0 or j == 0
            dict[i][j] = 0
        else
            if s[i] == t[j] then
                dict[i][j] = 1 + dict[i-1][j-1]
            else
                dict[i][j] = max{ dict[i][j-1], dict[i-1][j] }
```

`dict[m][n]` holds the solution (Practice with two strings)
Finding the longest common subsequence from the dictionary `dict`.

```
findLCS(i, j):
    if i == 0 or j == 0 return empty string
    if s[i] == t[j] return findLCS(i-1, j-1) + s[i];
    if dict[i][j] == dict[i-1][j]
        return findLCS(i-1, j)
    else
        return findLCS(i, j-1)
```

Invoke the above recursive function `findLCS(m, n)`.
Write the above iteratively.

```
// sol[i][j] = longest subsequence for strings of length i and j
for i = 0 to m
    for j = 0 to n
        if i == 0 or j == 0
            sol[i][j] = empty-string
        else
            if s[i] == t[j]
                sol[i][j] = sol[i-1][j-1] + s[i]
            else
                if dict[i][j] == dict[i][j-1]
                    sol[i][j] = sol[i][j-1]
                else
                    sol[i][j] = sol[i-1][j]
```

The construction of `dict` takes $O(m * n)$ time. The process of tracking the longest common subsequence also takes $O(m * n)$ time. Altogether, the runtime is $O(m * n)$.