Solution of Assignment #2
(Course: CS 401)

Problem 1 (50 points): A string $\mathcal{S}$ over an alphabet Σ is a concatenation of some symbols from Σ. For example, if $\Sigma = \{a, b, c\}$ then both *abacaabca* and *cbaaab* are strings over Σ.

For two strings $\mathcal{S}$ and $\mathcal{T}$, we say that $\mathcal{T}$ is a substring of $\mathcal{S}$ if $\mathcal{T}$ can be obtained from $\mathcal{S}$ by deleting one or more symbols from $\mathcal{S}$. For example, if $\mathcal{T} = cac$ and $\mathcal{S} = babcbaabbccca$ then $\mathcal{T}$ is a substring of $\mathcal{S}$ since

$\overbrace{\not{b}\not{a}\not{b}\,\boldsymbol{c}\,\not{b}\not{a}\,\boldsymbol{a}\,\not{b}\not{b}\not{c}\,\boldsymbol{c}\,\not{c}\not{a}}^{\mathcal{S}}$ is same as $\overbrace{cac}^{\mathcal{T}}$

Given two strings $\mathcal{S} = s_1 s_2 \dots s_n$ and $\mathcal{T} = t_1 t_2 \dots t_m$ over some alphabet Σ, the goal of this problem is to design a greedy algorithm that decides in $O(m+n)$ time if $\mathcal{T}$ is a substring of $\mathcal{S}$. For this purpose, answer the following questions.

(a) [20 points] Describe a greedy algorithm that, given two strings $\mathcal{S} = s_1 s_2 \dots s_n$ and $\mathcal{T} = t_1 t_2 \dots t_m$ over some alphabet Σ, does the following:

- decides if $\mathcal{T}$ is a substring of $\mathcal{S}$ and outputs a "yes" or "no" response accordingly, and
- if $\mathcal{T}$ is a substring of $\mathcal{S}$ then shows which symbols of $\mathcal{S}$ are deleted to make it same as $\mathcal{T}$.

It suffices to describe the algorithm in pseudo-codes as long as sufficient details are provided. Justify why your algorithm runs in $O(m+n)$ time.

(b) [30 points] Prove that your greedy algorithm works correctly. For this, you must show both of the following:

(b-1) [10 points] if your algorithm outputs "yes" then $\mathcal{T}$ is a substring of $\mathcal{S}$, and

(b-2) [25 points] if $\mathcal{T}$ is a substring of $\mathcal{S}$ then your algorithm outputs "yes".

Solution:

(a) We give a greedy algorithm that finds the first character in S that is the same as t_1, matches these two characters, then finds the first character after this in S that is the same as t_2, and so on. The algorithm looks as follows (comments are enclosed with $(*$ and $*)$):

$(*\ k_1, k_2, \dots$ denote the matches found so far $*)$
$(*\ i$ denotes the current position in S, j denotes the current position in $T\ *)$
$i \leftarrow 1$, $j \leftarrow 1$
while $i \leq n$ and $j \leq m$ do
    if $s_i = t_j$ then $k_j \leftarrow i$, $i \leftarrow i+1$, $j \leftarrow j+1$ else $i \leftarrow i+1$
endwhile
if $j = m+1$ then return the subsequence $k_1, \dots, k_m$
    else return "T is not a substring of S"

The running time is $O(n)$: one iteration through the while look takes $O(1)$ time, and each iteration increments i, so there can be at most n iterations.

(b) It is clear that the algorithm finds a correct substring if it finds any solution. It is harder to show that if the algorithm fails to find a substring, then no substring exists. Assume that T is the same as the substring $s_{\ell_1}, \ldots, s_{\ell_m}$ of S. We prove by induction on j that the algorithm will succeed in finding a substring and will have $k_j \leq \ell_j$ for all $j = 1, \ldots, m$. First consider $j = 1$. The algorithm lets k_1 be the first character in S that is the same as t_1, so we must have that $k_1 \leq \ell_1$. Now consider the case when $j > 1$. Assume that $j - 1 < m$ and assume by the induction hypothesis that the algorithm has $k_{j-1} \leq \ell_{j-1}$. The algorithm lets k_j be the first character in S after k_{j-1} that is the same as t_j if such a match exists. We know that ℓ_j is such a match and $\ell_j > \ell_{j-1} \geq k_{j-1}$. Thus $s_{\ell_j} = t_j$, and $\ell_j > k_{j-1}$. The algorithm finds the first such index, thus we get that $k_j \leq \ell_j$.

Problem 2 (30 points): This question is related to two claims made by Professor Smart, who has stated that he is smarter than the instructor and all the students in this class. We will examine his smartness by verifying the claims that he made.

Given an undirected, weighted graph G (with non-negative weights $w(u, v)$ for each edge $\{u, v\}$), and a constant $D > 0$, Professor Smart defines the graph G^{+D} as follows:

> G^{+D} is identical to G except we add the constant D to each edge weight.

(a) [15 points] Professor Smart claims that:

> a minimum spanning tree (MST) of G is always an MST of G^{+D}.

(b) [15 points] Professor Smart claims that:

> a shortest path between vertices s and t in G is always a shortest path between s and t in G^{+D}.

For each of the above two claims, your task is the following:

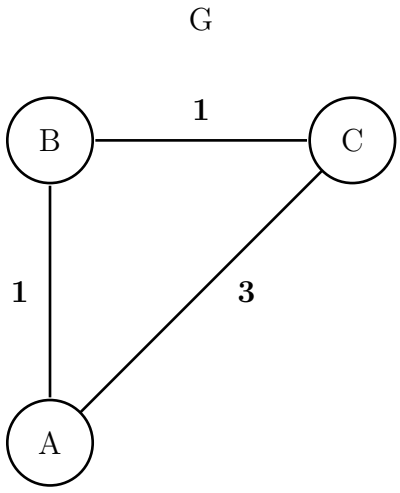
- If the claim is true, then provide a proof of the claim. This will show that Professor Smart was indeed smart.
- If the claim is false, provide a counter-example and explain why the counter-example shows that the claim is false. This will show that Professor Smart was not so smart after all.

Solution:

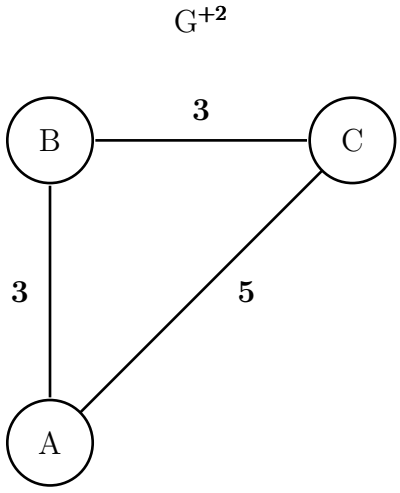
(a) Yes, the claim is true. To see why this is true, consider Kruskal's algorithm for finding an MST. In this algorithm, we consider the edges in non-decreasing order of weights. Since the

constant D is added to every edge, the ordering of edges by non-decreasing order of weights remains the same. So, Kruskal's algorithm will consider the edges in the same order and will generate the same MST.

(b) No, the claim is false. The following is a simple counter-example.



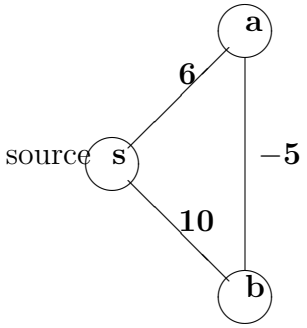
The shortest path between A and C is A–B–C of total weight 2



The shortest path between A and C is A–C of total weight 5

Problem 3 (20 points): Give a counter-example (graph) to show that Dijkstra's algorithm may not work correctly for a weighted graph with negative weights but with no cycles of total negative weight. Explain clearly why this is a counter-example; full credit will not be given without clear explanation.

Solution:
Consider the following weighted graph with no cycles of total negative weight:



The final result of Dijkstra's algorithm for shortest paths starting at vertex s are shown. Please note that the final values of $d[a]$ and $d[b]$ are wrong.