This assignment is **due on May 14**. All submitted work must be *done individually* without consulting someone else's solutions in accordance with the University's "Academic Dishonesty and Plagiarism" policies.

Problem 1. Our company has been a little overzealous in accepting work orders and we are wondering if we can complete all of them by the deadlines we agreed on. Every work order w_i is a tuple (d_i, t_i), where d_i is the deadline that we agreed on with our client, and t_i is the time needed to complete w_i. Time starts from $t = 0$. So, for example, if we have a the tuple $(4, 2)$, it needs to be completed by time $t = 4$, and it'll take 2 units of time to complete. All work orders are valid, meaning that $t_i \leq d_i$. For simplicity, we assume that all numbers are positive integers, that all t_i are distinct, and that all d_i are distinct. Since we have a small company, we can work on at most one work order at any given time.

Given this information, we have to find a feasible schedule (if one exists), i.e., an ordering of the work orders such that 1. every work order w_i is completed on or before its deadline d_i, and 2. the work done for w_i is t_i.

You are asked to evaluate two algorithms and determine whether they return a feasible schedule, if one exists. If none exists, we don't care what they return.

The strategy of ShortFirst is to sort the work orders by the time they take t_i. The strategy of MindTheDeadline is to sort the work orders by their deadline d_i. Both strategies process the work orders in their respective sorted order and keep track of the earliest time the next job can start (initially this is time $t = 0$) using *next*. The next job, say w_i, is then scheduled to start at *next* to ensure it doesn't overlap the previous jobs, and is performed for t_i units of time.

```
1: function ShortFirst(all w_i = (d_i, t_i))
2:     schedule ← [ ]
3:     next ← 0
4:     Sort by t_i in increasing order and renumber such that t_1 ≤ t_2 ≤ ... ≤ t_n
5:     for i ← 1; i ≤ n; i++ do
6:         schedule ← schedule ∪ (w_i, next)
7:         next ← next + t_i
8:     return schedule
```

```
1: function MindTheDeadline(all w_i = (d_i, t_i))
2:     schedule ← [ ]
3:     next ← 0
4:     Sort by d_i in increasing order and renumber such that d_1 ≤ d_2 ≤ ... ≤ d_n
5:     for i ← 1; i ≤ n; i++ do
6:         schedule ← schedule ∪ (w_i, next)
7:         next ← next + t_i
8:     return schedule
```

a) Show that ShortFirst doesn't always return a feasible schedule, even if one exists, by giving a counterexample.

b) Argue whether MINDTHEDEADLINE always returns a feasible schedule (if one exists) by either arguing its correctness (if you think it's correct) or by providing a counterexample (if you think it's incorrect).

Problem 2. You are given a simple undirected weighted graph $G = (V, E)$, a set of vertices $X \subset V$, and two vertices $s, t \in V \setminus X$. All weights are non-negative and you can assume that it takes $O(1)$ time to determine if a vertex is in X. The graph is given to you in adjacency list representation.

Your task is to design an algorithm that computes a shortest path between s and t in G, such that this path includes at most one vertex of the set X, if such a path exists. For full marks your algorithm needs to run in $O(m + n \log n)$ time.

a) Design an algorithm that solves the problem.

b) Briefly argue the correctness of your algorithm.

c) Analyse the running time of your algorithm.

Problem 3. We are given a simple undirected graph of maximum degree d and $d + 1$ different tokens (assume an infinity number of each token, so you don't need to worry about running out). For ease of description, our tokens are numbered 0 to d. We aim to place a token on every vertex of the graph in such a way that for every edge of the graph, its two endpoints have different tokens on them. The graph is given to you in adjacency list representation.

Your task is to design an algorithm that takes a graph G and the set of tokens T as input and returns a valid placement of the tokens on the graph, if one exists. For full marks your algorithm needs to run in $O(n + m)$ time.

a) Design an algorithm that solves the problem.

b) Briefly argue the correctness of your algorithm.

c) Analyse the running time of your algorithm.

Advice on how to do the home work

- Assignments should be typed and submitted as pdf (no handwriting)
- When designing an algorithm or data structure, it might help you (and us) if you briefly describe your general idea, and after that you might want to develop and elaborate on details. If we don't see/understand your general idea, we cannot give you points for it.
- Be careful with giving multiple or alternative answers. If you give multiple answers, then we will give you marks only for "your worst answer", as this indicates how well you understood the question.
- Some of the questions are very easy (with the help of the lecture notes or book). You can use the material presented in the lecture or book without proving it. You do not need to write more than necessary (see comment above).
- When giving answers to questions, always prove/explain/motivate your answers.
- When giving an algorithm as an answer, the algorithm does not have to be given as (pseudo-)code.
- If you do give (pseudo-)code, then you still have to explain your code and your ideas in plain English.
- Unless otherwise stated, we always ask about worst-case analysis, worst case running times etc.
- As done in the lecture, and as it is typical for an algorithms course, we are interested in the most efficient algorithms and data structures.
- If you use further resources (books, scientific papers, the internet,...) to formulate your answers, then add references to your sources and show clearly that you understand what you're referencing by explaining the approach as you would normally.
- If you refer to a result in a scientific paper or on the web you need to explain the results to show that you understand the results, and how it was proven.