Worth: 12% **Due:** By 8:59pm on Tuesday 24 February

Remember to write the *full name* and *student number* of *every group member* prominently on your submission.

---

*Please read and understand the policy on Collaboration given on the Course Information Sheet. Then, to protect yourself, list on the front of your submission **every** source of information you used to complete this homework (other than your own lecture and tutorial notes). For example, indicate clearly the **name** of every student from another group with whom you had discussions, the **title and sections** of every textbook you consulted (including the course textbook), the **source** of every web document you used (including documents from the course webpage), etc.*

*For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks **will** be deducted for incorrect or ambiguous use of notation and terminology, and for making incorrect, unjustified, ambiguous, or vague claims in your solutions.*

---

1. (a) Give an efficient algorithm that takes the following inputs:
   - $G = (V, E)$, a connected undirected graph,
   - $w : E \to \mathbb{Z}^+$, a weight function for the edges of G,
   - $T \subseteq E$, a minimum spanning tree of G,
   - $e_1 = \{u, v\} \notin E$ (for $u, v \in V$), an edge not in G,
   - $w_1 \in \mathbb{Z}^+$, a weight for e_1,

   and that outputs a minimum spanning tree T_1 for the graph $G_1 = (V, E \cup \{e_1\})$ with $w(e_1) = w_1$.

   For full marks, your algorithm must be more efficient than computing a MST for G_1 from scratch. Justify that this is the case by analysing your algorithm's worst-case running time.

   Finally, write a detailed proof that your algorithm is correct. (Note that this argument of correctness will be worth at least as much as the algorithm itself.)

   (b) Give an efficient algorithm that takes the following inputs:
   - $G = (V, E)$, a connected undirected graph,
   - $w : E \to \mathbb{Z}^+$, a weight function for the edges of G,
   - $T \subseteq E$, a minimum spanning tree of G,
   - $e_0 \in E$, an edge in G,

   and that outputs a minimum spanning tree T_0 for the graph $G_0 = (V, E - \{e_0\})$, if G_0 is still connected—your algorithm should output the special value NIL if G_0 is disconnected.

   For full marks, your algorithm must be more efficient than computing a MST for G_0 from scratch. Justify that this is the case by analysing your algorithm's worst-case running time.

   Finally, write a detailed proof that your algorithm is correct. (Note that this argument of correctness will be worth at least as much as the algorithm itself.)

2. During the renovations at Union Station, the work crews excavating under Front Street found veins of pure gold ore running through the rock! They cannot dig up the entire area just to extract all the gold: in addition to the disruption, it would be to expensive. Instead, they have a special drill that they can use to carve a single path into the rock and extract all the gold found on that path. Each crew member gets to use the drill once and keep the gold extracted during their use. You have the good luck of having an uncle who is part of this crew. What's more, your uncle knows that you are studying computer science and has asked for your help, in exchange for a share of his gold!

The drill works as follows: starting from any point on the surface, the drill processes a block of rock 10cm × 10cm × 10cm, then moves on to another block 10cm below the surface and connected with the starting block either directly or by a face, edge, or corner, and so on, moving down by 10cm at each "step". The drill has two limitations: it has a maximum depth it can reach and an initial hardness that gets used up as it works, depending on the hardness of the rock being processed; once the drill is all used up, it is done even if it has not reached its maximum depth.

The good news is that you have lots of information to help you choose a path for drilling: a detailed geological survey showing the hardness and estimated amount of gold for each 10cm × 10cm × 10cm block of rock in the area. To simplify the notation, in this homework, you will solve a two-dimensional version of the problem defined as follows.

Input: A positive integer d (the initial *drill hardness*) and two $[m \times n]$ matrices H, G containing non-negative integers. For all $i \in \{1,\ldots,m\}, j \in \{1,\ldots,n\}$, $H[i,j]$ is the *hardness* and $G[i,j]$ is the *gold content* of the block of rock at location i,j (with $i = 1$ corresponding to the surface and $i = m$ corresponding to the maximum depth of the drill).

There is one constraint on the values of each matrix: $H[i,j] = 0 \Rightarrow G[i,j] = 0$ (blocks with hardness 0 represent blocks that have been drilled already and contain no more gold).

Figure 1 below shows the general form of the input. Figure 2 shows a sample input.

Output: A drilling path $j_1, j_2, \ldots, j_\ell$ for some $\ell \leqslant m$ such that:

- $\boxed{\mathbf{1 \leqslant j_k \leqslant n \text{ for } k = 1, 2, \ldots, \ell \text{ (each coordinate on the path is valid);}}}$
- $j_{k-1} - 1 \leqslant j_k \leqslant j_{k-1} + 1$ for $k = 2, \ldots, \ell$ (each block is underneath the one just above, either directly or diagonally);
- $H[1, j_1] + H[2, j_2] + \cdots + H[\ell, j_\ell] \leqslant d$ (the total hardness of all the blocks on the path is no more than the initial drill hardness);
- $G[1, j_1] + G[2, j_2] + \cdots + G[\ell, j_\ell]$ is maximum (the path collects the maximum amount of gold possible).

Follow the dynamic programming paradigm given in class (the "five step" process) to give a detailed solution to this problem. Make sure to keep a clear distinction between each of the steps, and to explain what you are doing and why at each step—you will **not** get full marks if your answer contains only equations or algorithms with no explanation or justification.

$H[1,1], G[1,1]$	$H[1,2], G[1,2]$	$\cdots$	$H[1,n], G[1,n]$
$H[2,1], G[2,1]$	$H[2,2], G[2,2]$	$\cdots$	$H[2,n], G[2,n]$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$H[m,1], G[m,1]$	$H[m,2], G[m,2]$	$\cdots$	$H[m,n], G[m,n]$

Figure 1: General form of the input matrix.

2,2	**2,7**	0,0	2,3	1,8
2,2	**0,0**	1,2	2,0	1,1
1,4	1,1	**2,6**	2,1	3,8

Figure 2: Sample input with optimum path shown in **bold** (for $d = 4$).

3. Every year in April, the Faculty of Arts & Science is responsible for more than 1200 different final examinations! This year, the faculty faces a crisis: the person in charge of finding *Chief Presiding Officers* (CPOs) for all the examinations is retiring. Luckily for them, your sister-in-law works in

that office—you have a surprisingly well-connected family—and she approaches you to help solve this problem. (There's no gold to reward you with this time, though!)

The office in charge of assigning CPOs to examinations works as follows.

- The office has a list of every examination taking place with the date and time of each examination (where time is one of "morning," "afternoon" or "evening"). Each combination of date and time (like "the morning of April 14") is called an *examination period*.
- For every CPO, the office has a list of examination periods when the CPO is available. A CPO cannot be assigned to an examination period when he/she is not available.
- For every CPO, the office knows the maximum number of examinations that the CPO can invigilate. This will usually (but not necessarily) be smaller than the number of examination periods when the CPO is available. And it can be different for different CPOs.
- No CPO can invigilate more than two different examinations on the same day.
- For historical reasons, the process of assigning CPOs to examinations happens in two separate phases.
  - First, CPOs are assigned to a *pool* for each examination period. Each pool should contain 10% more CPOs (rounded up) than the number of examinations taking place during the period (to cover cases when CPOs become unavailable at the last minute, due to circumstances outside their control). For example, if there are 6 examinations scheduled on the morning of April 14, the pool for this period should contain $\lceil 6 \times 1.1 \rceil = 7$ CPOs.
  - Second, a few hours before each examination period, CPOs from the corresponding pool are assigned to actual examinations, at random. This is a security measure to make it impossible for students to try to bribe CPOs, since a CPO will not know ahead of time which examination he/she is invigilating.

  It is the first phase only (the more difficult one) that the office needs help with.

Write a polynomial time algorithm that assigns CPOs to *examination periods*. More precisely, your algorithm takes as inputs the data specified above and it outputs a list of examination periods for each CPO that satisfies the constraints above. The goal is to assign enough CPOs to every examination period (as described above): your algorithm should report if this is not possible. Note that it is perfectly fine if none of the CPOs are assigned their maximum number of examinations, as long as every examination period gets enough CPOs.

Please provide an English explanation of your algorithm, either as a short paragraph to describe the main idea, or as comments throughout your pseudocode. You should *call* algorithms from class when appropriate (instead of pasting them into your code)

Then, analyse the worst-case running time of your algorithm. **Also justify that it is correct.** Please quote results from class where appropriate (rather than re-deriving them). And be careful with the details: use the correct names and labels for any algorithm or theorem that you use.