**Due Friday 14th April at 5pm Sydney time**

In this assignment we apply dynamic programming and associated graph algorithms. There are *three problems* each worth 20 marks, for a total of 60 marks. Partial credit will be awarded for progress towards a solution. We'll award one mark for a response of "one sympathy mark please" for a whole question, but not for parts of a question.

Any requests for clarification of the assignment questions should be submitted using the Ed forum. We will maintain a FAQ thread for this assignment.

For each question requiring you to design an algorithm, you *must* justify the correctness of your algorithm. If a time bound is specified in the question, you also *must* argue that your algorithm meets this time bound. The required time bound always applies to the *worst case* unless otherwise specified.

You must submit your response to each question as a separate PDF document on Moodle. You can submit as many times as you like. Only the last submission will be marked.

Your solutions must be typed, *not* handwritten. We recommend that you use LaTeX, since:

- as a UNSW student, you have a free Professional account on Overleaf, and
- we will release a LaTeX template for each assignment question.



Other typesetting systems that support mathematical notation (such as Microsoft Word) are also acceptable.

Your assignment submissions must be your own work.



- You may make reference to published course material (e.g. lecture slides, tutorial solutions) without providing a formal citation. The same applies to material from COMP2521/9024.
- You may make reference to either of the recommended textbooks with a citation in any format, *except* that you may not use network flow algorithms not presented in lectures.



- You may reproduce general material from external sources in your own words, along with a citation in any format. 'General' here excludes material directly concerning the assignment question. For example, you can use material which gives more detail on certain properties of a data structure, but you cannot use material which directly answers the particular question asked in the assignment.
- You may discuss the assignment problems privately with other students. If you do so, you must acknowledge the other students by name and zID in a citation.
- However, you must write your submissions entirely by yourself.
  - Do not share your written work with anyone except COMP3121/9101 staff, and do not store it in a publicly accessible repository.
  - The only exception here is UNSW Smarthinking, which is the university's official writing support service.

Please review the UNSW policy on plagiarism. Academic misconduct carries severe penalties.

Please read the Frequently Asked Questions document, which contains extensive information about these assignments, including:

- how to get help with assignment problems, and what level of help course staff can give you
- extensions, Special Consideration and late submissions
- an overview of our marking procedures and marking guidelines
- how to appeal your mark, should you wish to do so.

## Question 1

Given a strictly increasing sequence $A$ whose entries are positive integers $[a_1, \ldots, a_n]$, a sub-sequence of $A$ is defined by removing zero or more elements from $A$ while preserving the original order. For example, $[1, 13, 21]$ and $[1, 2, 3, 13, 18, 21, 71]$ are both sub-sequences of $[1, 2, 3, 13, 18, 21, 71]$.

A sequence $x_1, x_2, ..., x_m$ is called *beautiful* if:

- $m \geq 3$, that is, the number of elements in the sequence is greater or equal to 3.
- $3x_i + 5x_{i+1} = x_{i+2}$ for all $i \leq n - 2$.

In the example above, the beautiful sub-sequences are:

- $[1, 2, 13]$,
- $[1, 3, 18]$,
- $[2, 3, 21]$,
- $[2, 13, 71]$ and
- $[1, 2, 13, 71]$.

**1.1** **[3 marks]** Suppose $[\ldots, a_j, a_k]$ is a beautiful sub-sequence of $A$. Given the indices $j$ and $k$ of the last two entries of the sub-sequence, design an algorithm which runs in $O(\log n)$ time and computes the index in $A$ of the third last entry of the sub-sequence.

Your algorithm should find the index $i$ of the entry $a_i$ which precedes $a_j$ and $a_k$ in the sequence, assuming that such an $i$ exists.



**1.2** **[12 marks]** Design an algorithm which runs in $O(n^2 \log n)$ time and computes the length of the longest beautiful sub-sequence of $A$.



We believe this problem can be solved in $O(n^2)$ time. A solution satisfying this constraint will earn one bonus mark for the course.

**1.3** **[3 marks]** Design an algorithm which runs in $O(n \log n)$ additional time and lists the entries of the longest beautiful sub-sequence of $A$.

'Additional' here means after the execution of your algorithm for 1.2.

If there are two or more equal longest beautiful sub-sequences, your algorithm should produce any one of them.

## Question 2

You are in a warehouse represented as a grid with $m$ rows and $n$ columns, where $m, n \geq 2$. You are initially located at the top-left cell $(1, 1)$, and there is an exit at the bottom-right cell $(m, n)$. There are certain cells that contain boxes which you can not move through. The grid is given as a 2D array $B[1..m][1..n]$ where $B[i][j]$ is TRUE if cell $(i, j)$ contains a box and FALSE otherwise.

**2.1** **[6 marks]** Occupational health and safety regulations specify that it must be possible to reach the exit from your starting point by making only two kinds of moves: down one cell or right one cell. Design an algorithm that runs in $O(mn)$ time and determines whether the warehouse layout meets this requirement.

Note: there is both a DP and a non-DP approach to this question. Both are eligible for full marks in this part, but the DP approach naturally lends itself to be extended to Question 2.2. If you choose the non-DP approach here, you will likely need to put in more work to get full marks in Question 2.2.

**2.2** **[3 marks]** Unfortunately, some warehouse layouts do not meet this requirement. You have been asked to remove some boxes to ensure that the requirement is met, but wish to remove as few as possible to make efficient use of warehouse space. Design an algorithm that runs in $O(mn)$ time and determines the smallest number of boxes that must be removed to meet the requirement.

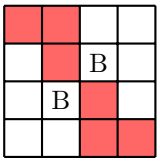


The fire alarm has gone off, so you have to make your escape! Fortunately, this warehouse passed the safety inspection, so you know there is a path to the exit that only involves moving down or right one cell at a time. However, you also want to ensure you can make a speedy escape, so you must take **exactly one** shortcut by moving diagonally: one cell down and right at the same time.





For example, the red path through this warehouse includes a shortcut from $(2, 2)$ to $(3, 3)$.




**2.3** **[2 marks]** Show that any warehouse passing the safety requirement from 2.1 has a path from $(1, 1)$ to $(m, n)$ that includes a shortcut.

**2.4** **[9 marks]** Each cell of the warehouse that does not contain a box has a particular hazard rating $H[i][j]$, and you want to minimise the sum of the hazard ratings on your path to the exit.

For example, the red path through this warehouse layout has a total hazard rating of 13, takes exactly one shortcut, and is the minimal path for this particular warehouse.

| | | | |
|---|---|---|---|
| **2** | **2** | 6 | 1 |
| 3 | **4** | B | 1 |
| 2 | B | **2** | 2 |
| 1 | 4 | **2** | **1** |

Design an algorithm that runs in $O(mn)$ time and determines the minimum total hazard rating you can achieve.

## Question 3

You have been asked to perform $n$ complex calculations $c_1, \ldots, c_n$, where each calculation $c_i$ requires $r_i$ bytes of RAM to store the result. You can perform the calculations *in any order*.

Some calculations depend on the result of others. These dependencies are represented as a directed graph $G = (V, E)$, where $E$ contains an edge $c_j \to c_i$ if the result of $c_j$ has to be in RAM in order to calculate $c_i$.

Let $\text{pred}(i)$ denote the set of computations that $c_i$ depends on, that is,

$$\text{pred}(i) = \{j : (c_j \to c_i) \in E\}.$$

**3.1** **[3 marks]** Explain why it is impossible to perform all $n$ calculations unless the graph $G$ is acyclic.

**3.2** **[10 marks]** To perform calculation $c_i$, we first have to collate the results of $\text{pred}(i)$, which takes

$$\sum_{j \in \text{pred}(i)} r_j$$

seconds. It then takes $r_i^2$ seconds to compute the result.



On a sequential computer which can only perform one calculation at a time, it takes



$$\sum_{i=1}^{n} \left( r_i^2 + \sum_{j \in \text{pred}(i)} r_j \right)$$

seconds to determine all results. However, we have a massively parallel computer that can perform an unlimited number of calculations at the same time.



Design an algorithm that runs in $O(n+m)$ time and determines the minimum amount of time required to perform all $n$ calculations on the parallel computer, where $m$ is the number of dependencies, i.e. the number of edges in the graph.

**3.3** **[7 marks]** We now have access to a supercomputer which can compute results instantly. If we choose to use the supercomputer for calculation $c_i$, it takes only

$$\sum_{j \in \text{pred}(i)} r_j$$

time to collate the previous results, without the additional $r_i^2$ seconds to compute the result.

The supercomputer is very expensive to run, so it can be used at most $s$ times.

Design an algorithm that runs in $O(s(n+m))$ time and determines the minimum amount of time required to compute all $n$ results using the supercomputer for at most $s$ calculations, and the parallel computer for all other calculations.