Due Friday 10th of November at 6pm Sydney time (week 9)

In this assignment we will apply dynamic programming. 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.
- 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 *OurExperience*

You are collecting survey results from k students over n days. Each day, all k students answer the survey by providing a *real* number between 0 and 10 representing their stress level. For each student, their survey responses over the n days are given in an array $S_i[1..n]$.

You want to combine all k survey responses to report to your boss how the students are feeling. You are going to make a report which uses one of the responses from each day, and you want to pick-and-choose responses so that students appear to be emotionally stable. You have designed a heuristic to do so:

> Choose a sequence of survey responses $R[1..n]$, where each $R[i]$ is the response from one of the students on day i, minimising the change between each response in the sequence. That is, choose a sequence R where each $R[i]$ is one of $\{S_1[i], S_2[i], \dots, S_k[i]\}$ minimising the *fluctuation*
>
> $$f = \sum_{i=2}^{n} |R[i-1] - R[i]|\,.$$

For example, with $n = 4$ and $k = 3$, if you measure the results

$$S_1 = [2, 5, 8, 9.9], \qquad S_2 = [5, 2.5, 1, 4], \qquad S_3 = [10, 3, \pi^2, 7],$$

then the optimal sequence is $R = [5, 5, 8, 7]$ with

$$f = |5 - 5| + |5 - 8| + |8 - 7| = 4,$$

obtained by taking the measurements from sensors $2, 1, 1, 3$.

1.1 **[6 marks]** Consider this attempt to solve the problem with a greedy algorithm:

> Start with $S_1[1]$, then for each subsequent day choose the survey response closest to the previous one. Repeat by starting at each $S_2[1], S_3[1], \dots, S_k[1]$, and of these, choose the sequence with the minimum fluctuation.

Show that this algorithm does not solve the problem correctly by giving a counterexample. You must provide n, k, the k sequences of survey responses, the answer (R and f) produced by the greedy algorithm, and the correct answer. Your example must have $n \leq 4$ and $k \leq 3$.

1.2 **[14 marks]** Design an $O(nk^2)$ algorithm to find the minimal fluctuation of an optimal sequence R, and the sequence of survey responses used to achieve this fluctuation. If there are multiple optimal sequences, your algorithm can find any one of them.

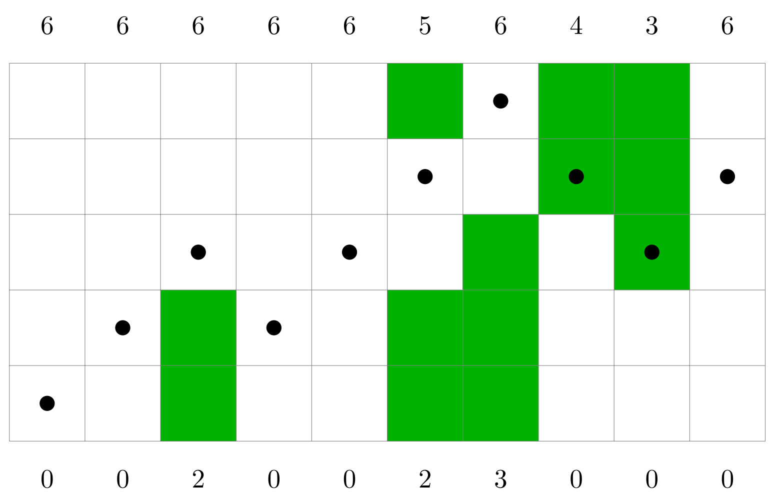
> Answers that determine only the minimal fluctuation (without finding the sequence of responses) will receive at most 10 marks for this part.

Question 2 *Crappy Bird*

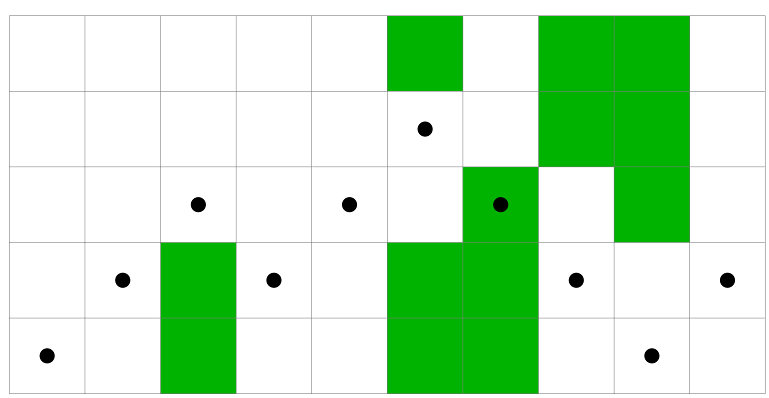
Crappy Bird is a game where you control a bird starting at $(1, 1)$ in a grid with m rows of n columns (where $m \geq 2$). Some columns j ($1 \leq j \leq n$) have pipes from the bottom of the screen to row $\mathsf{PipeUp}(j)$, and some have pipes from the top of the screen to row $\mathsf{PipeDown}(j)$. If a column j has no pipe from the bottom then $\mathsf{PipeUp}(j) = 0$, and if it has no pipe from the top then $\mathsf{PipeDown}(j) = m + 1$. The goal is to reach row n while hitting as few pipes as possible. If you exit the screen by flying too low or too high, you lose the game immediately!

Each step of the game, the bird moves one column to the right. Additionally, if you tap the screen the bird moves one row up, otherwise it falls one row down. That is, if the bird is at (x, y) at the some step, then it can be at $(x + 1, y + 1)$ or $(x + 1, y - 1)$ in the next step.

For example, this is a valid path which hits two pipes. The PipeUp and PipeDown values are shown below and above the grid respectively.



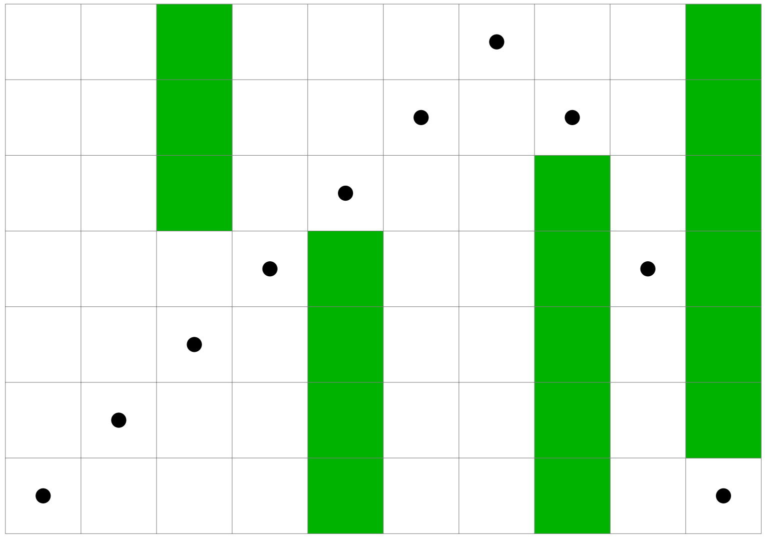
A better path is achieved by going down from $(6, 4)$ instead of up, hitting only one pipe.



2.1 **[10 marks]** Design an algorithm that runs in $O(mn)$ time, and determines the fewest pipes the bird can run into without exiting the screen.

2.2 **[7 marks]** The developers have realised that the game's implementation of gravity is entirely unrealistic! To fix it, they have patched the game to include acceleration for downwards movement. When the bird begins falling (after a sequence of upwards movements), it first falls 1 unit downwards. If it falls again in the next step, it falls 2 units, then 3, 4, and so on. That is, the kth consecutive time the bird falls, it moves from (x, y) to $(x + 1, y - k)$.

For example, the bird may take this path:



Note that no pipes are hit, as at each step the bird is in a cell that does not coincide with a pipe. **The movement between positions is not considered.**

Design an algorithm that runs in $O(m^2n)$ time, and determines the fewest pipes the bird can run into.

2.3 [3 marks] Describe how the algorithm for part 2.2 can be improved to $O(nm\sqrt{m})$. You do not need to restate the entire algorithm - just describe and justify how you would change the algorithm to achieve this.

You may choose the skip this part and instead give an $O(nm\sqrt{m})$ algorithm for part 2. If you do, your answer to 2.2 will be marked for both parts.

Question 3 *Dance!*

In the arcade game *Dance Dance Revolution* (DDR), players stand on a stage and hit arrows as they scroll across the screen. More specifically, a sequence of n arrows (←, ↑, →, ↓) will scroll across the screen, and as each arrow hits the top of the screen, the player must stand on the corresponding arrow on the stage.

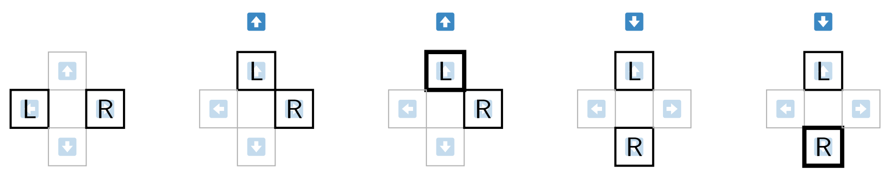
We play a variant of DDR, aptly named *Don't Dance Revolution* (DDR2), where the goal is to play the game like DDR but move as little as possible. The game plays like DDR except when an arrow reaches the top of the screen and the player already has a foot on the correct arrow, then the player is awarded one lazy point. If neither foot is on the correct arrow, then the player must move *exactly* one foot from its current location to the correct arrow on the platform.

Unfortunately, the game is a bit unforgiving: any wrong move will cause the player to lose the game and *all* of their lazy points. Wrong moves include:

- Failing to step on the correct arrow.
- Moving more than one foot at any given time.
- Moving either foot when the player is already stepping on the correct arrow.

You are given a sequence A of n arrows. Assume that your left foot starts on ← and your right foot starts on →, and you have memorised the entire sequence of arrows.

For example, consider the following sequence: ↑ ↑ ↓ ↓. We can earn up to two lazy points as follows:



Start game! *Lazy point!* *Lazy point!*

3.1 **[4 marks]** Show that it is always possible to earn at least $\lfloor n/4 \rfloor$ lazy points during a round of DDR2.

3.2 **[16 marks]** Design an $O(n)$ algorithm to determine the maximum number of lazy points you can earn in a round of DDR2.