CSC3100 Assignment 2 REPORT

Xue Zhongkai 122090636

Problem 1

This problem is to find the winner who sends out nothing, but receives notes from all others. The input module will illustrate the transaction in the form of the matrix $M[i][j]$, and I am required to find the winner.

Intuitively, we could go through all the candidates and check if it satisfies the condition one by one. However, it is of the complexity $O(N^2)$ if the whole matrix is traversed, and will cause TLE issue.

So the key approach falls on how to find an $O(N)$ algorithm to find the winner. Hence I initialized a candiate by 0, and set the new candiate to be the receiver of the candiate, and traverse all the possible candidates . As the restriction is that there could be only one winner, in the next loopreturn -1 (no winner) if the candidate sends a note to others himself.

The other things is that reading a whole line from the buffer will be more efficient for Python. Also I am worried about the matrix storation of $M[i][j]$ could be inefficient for large-scale data, for which I flatten the matrix, and store it in the form of $M[i * k + j]$ as a list. Finally it passed all the tests.

Problem 2

This problem can be solved as follows: Given an integer array *nums* and an integer *k*, a sliding window moves from the leftmost to the rightmost of the array, advancing one position at a time. Take in an integer array *nums*, returns an array where each element represents the maximum value in the sliding window at each move.

For each element *i* in the entire array *nums*:

If the deque *dq* is not empty and the index corresponding to the front of the deque has exceeded the current sliding window's range $i - k$, remove the front element; If the deque *dq* is not empty and the value corresponding to the element at the back of the deque is less than the current element in the array, remove the back element .

Finally, add the current index *i* to the deque *dq*. If the current index *i* is greater than or equal to the size of the sliding window *k - 1*, add the maximum value of the current sliding to the result list *ans*.

This has been taught in the tutorial and could be solved similarly.

Problem 3

The characteristic of "last-in, first-out" (LIFO) with immediacy reminds me of that of a stack. In the process, I need to maintain a certain order and compare the current element with previous ones during processing. If the current element is smaller than some previous elements, it's necessary to remove those previous elements from the stack. This characteristic of stacks make it convenient to maintain a decreasing sequence and easily remove elements that do not satisfy the requirements.

This is exactly what I do:

Iterate through the array and push each element onto the stack one by one. Before pushing an element onto the stack, check whether the top element of the stack is greater than the current element. If it is, pop the top until this condition is met or *m* becomes 0. If the value of *m* is still greater than 0, elements are popped from the stack until that is satisfied.

With simple helper methods, the elements in the stack are presented as a string in order, and leading zeros are removed. It returns 0 if the result is empty.