Editorial

quangvn2508

February 11, 2020

In this editorial, I would like to provide my approach for each of the questions. These solutions are not necessarily the only or the best and optimised solution. If you have a better solution and want to share please let us know and it will be updated.

The solutions and test cases can be found in the same folder.

Note that it is not possible to click on links if you are viewing through GitHub's editor, and hence it is recommended to download it directly.

1 Question 1: String evaluation [problem link]

Setter: quangvn2508
Solution by quangvn2508

- Time complexity: O(n)

Straightforward, go through each of character in the string (S) and add the ASCII value of that character to the total sum.

Algorithm 1 Problem 1

```
1: $sum \leftarrow 0$
2: for all character c in S do
3:     $sum \leftarrow sum$ + ASCII value of c
4: end for
5: print $sum$ to standard output
```

2 Question 2: Skyfall [problem link]

Setter: quangvn2508
Solution by quangvn2508

- Time complexity: $O(n \log n)$

The first step is to note that there is no guarantee that the array is already sorted (the sample test cases are). Therefore, I will first sort the array in decreasing order, using the sort function in standard library, which has time complexity $O(n \log n)$.

The idea is, as we pass through the sorted array, at each position, we need to calculate the total number of floors that the sky need to destroy before we reach the current height. If the total number of floors is more than P of the sky then the sky is stopped. Consider the below algorithm:

Let us define some symbols used in the algorithm:

- height is the array denoted to the height of each building (remember that we have sorted the array)
- height[i] is the height of building at position i ($0 \leq i \leq n-1$) and height[i] > height[i + 1].

Algorithm 2 Problem 2

```
current_height ← height[0]
current_floors ← 0
for i = 1, 2, ..., n − 1 do
    change_of_height ← current_height − height[i]
    increase_of_floors ← change_of_height ∗ i
    if current_floors + increase_of_floors ≥ P then
        print current_height − ⌈(P − current_floors)/i⌉
    else
        current_floors ← current_floors + increase_of_floors
        current_height ← height[i]
    end if
end for
print −1
```

Consider the sample test case:

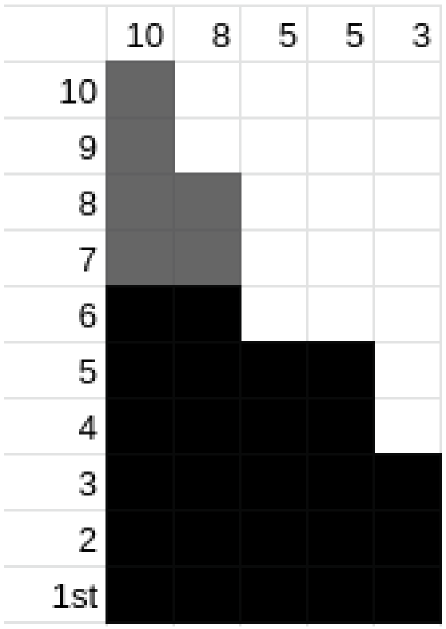


Figure 1: Sample test case

At position 0, the total floor that sky needs to destroy before reaching this height is 0 (since it is the highest building).

At position 1, the total floors is 2, as in order to reach height 8, the sky need to destroy 2 floors of the first building. Therefore, current_floors = change_of_height * $i = 2 * 1$. Note that i also denotes the number of buildinsg before building i.

Moving on to position 2, the total floors to add is 6. Although the difference of height is 3, there are two buildings before position 2. In this example, $P = 6$, therefore, current_floors + increase_of_floors = 2+6 $> P = 6$.

Now we need to work out the floor where current_floors + increase_of_floors is just over P. At height 8, the sky has destroyed 2 floors, therefore the remaining is 4. To work out how deep the sky will go, we use the width (i) and divide the remaining value; then the result will be 2, which mean the sky will stop after destroying 2 floors below 8.

We need to be careful when the result is not an integer. For example, consider sample test 2 where P is 7 instead of 6. At the last step, the result is 2.5, which indicates that the sky stopped between floor 6 and 5. However, we are asked to report the number where the sky has stopped. Hence the result needs to be rounded up.

Comment: since this algorithm goes through the array once, the time complexity is O(n). However, the time complexity overall is $O(n \log n)$ because of sorting.

3 Question 3: kth digit of nth Fibonacci number [problem link]

Setter: quangvn2508
Solution by quangvn2508

- Time complexity: O($n * k$)

Look at the constraints, we can see that the number can be very large (normal sequence starts at 0 note that we have $F_{10000} > 10^{2000}$).

As I have mentioned in the contest analysis, this question is unfair for those who did use Python. However, some participants (possibly all) exploited this language too much. For example, if the test case ask for the 2^{nd} digit of the F_{10000} then after compute the F_i, the result should be modulo with 100 to save memory.

For those who did not know that python can handle large numbers then this section is for you. The solution I have in mind is to use array as an representation for the numbers and perform addition like we did in elementary school.

Let's define some symbol used in the algorithm:

- F_i, F_{i-1}, F_{i-2} are arrays of size k+1
- $F_i[j]$ is the j^{th} digit from the least significant.

Algorithm 3 Problem 3

```
1:  while $n > 2$ do
2:    for $j = 0, 1, 2, ..., k-1$ do
3:      $F_i[j] \leftarrow F_{i-1}[j] + F_{i-2}[j]$
4:      if $F_i[j] > 9$ then
5:        $F_i[j+1] \leftarrow \lfloor F_i[j]/10 \rfloor$
6:        $F_i[j+1] \leftarrow F_i[j] \pmod{10}$
7:      end if
8:      $F_{i-2} \leftarrow F_{i-1}$
9:      $F_{i-1} \leftarrow F_i$
10:     RESET $F_i$
11:   end for
12: end while
13: print $F_{i-1}[k-1]$
```

If the array size is k then we might have problem, as if k^{th} digit is overflow (> 9) then we will have run-time error. By setting the size of array to$k$, we can avoid more complicated code. The logic is trivial and hence will not be mentioned here.

4 Question 4: Influenced sum [problem link]

Setter: quangvn2508
Solution by quangvn2508

- Time complexity: average O(n), worst O(n^2)

The influence of a node depends on the longest distance from start node (S). Therefore, the obvious approach is to traverse the graph from (S) to find the longest distance (D), then traverse a second time to calculate the influenced value.

However, it is possible to calculate the influenced sum by performing breadth-first-search once. The idea is to find the sum of labels of all nodes with the same distance from (S). We can do this since all node have the same shortest distance (d) from (S) will have influence (F) and their sum$_\mathrm{d}$ is

$$sum_d = n_1 * F + n_2 * F + ... + n_i * F = (n_1 + n_2 + ... + n_i) * F$$

After calculating all of $sum_0, sum_1, ..., sum_D$, wecan calculate the influenced sum by this equation

$$sum = sum_0 * (D + 1) + sum_1 * (D) + sum_2 * (D - 1) + ... + sum_{D-1} * 2 + sum_D * 1$$

Let define some symbols used in this algorithm:

- *Sum_array* will holds sum of node with the same distance.
- *Sum_array*[i] will contains sum_i as mentioned above.
- *QNode* is a queue of nodes' label.

Algorithm 4 Problem 4: BFS using queue

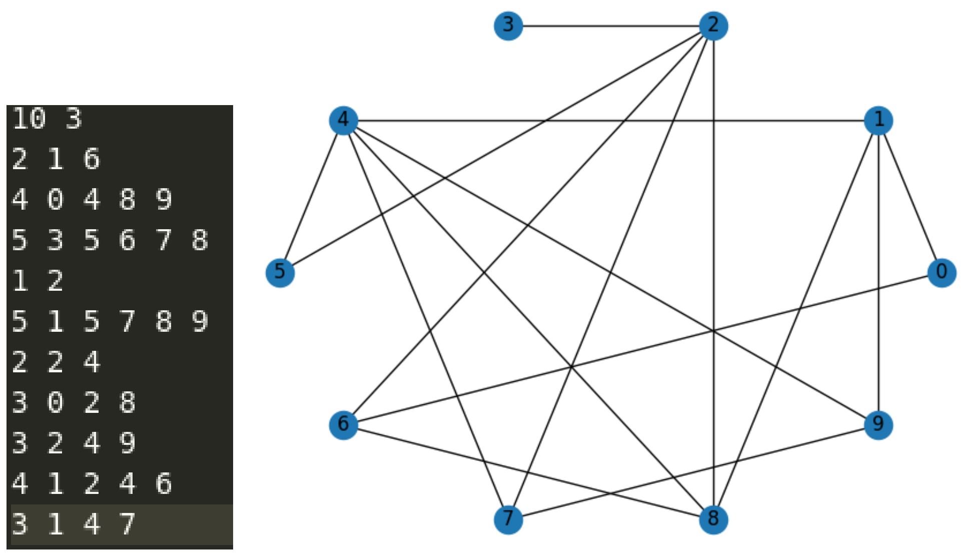
```
1: PUSH start node $S$ to $QNode$
2: while QNode not empty do
3:   $s \leftarrow$ size of $QNode$
4:   $sum_i \leftarrow 0$
5:   for $j = 1, 2, ..., s$ do
6:     $current\_node \leftarrow$ pop first element in $QNode$ $sum_i \leftarrow sum_i + current\_node$
7:     for unvisited nodes which is adjacent to $current\_node$ do
8:       mark node as visited
9:       PUSH node to QNode
10:    end for
11:  end for
12:  ADD $sum_i$ to back of $Sum\_array$
13: end while
```

Algorithm 5 Problem 4: Calculating the influenced sum

```
1: $max\_influence \leftarrow$ size of $Sum\_array$
2: $influence\_sum \leftarrow 0$
3: for $i = 0, 1, ..., max\_influence - 1$ do
4:   $influence\_sum \leftarrow influence\_sum + (max\_influence - i) * Sum\_array[i]$
5: end for
6: print $influence\_sum$
```

Note that every time we repeat the loop on line 2, the distance from start node (S) increases by 1 since the number of pop we perform in one while loop is equal to the size of queue at the beginning. Each node is added and removed from the *QNode* once, therefore, BFS will have average time complexity of O(n). However, it is difficult to analyse the number of repetitions for the for loop on line 7. The worst-case scenario occurs when the graph is a completed graph, since for each node, we need to go through their adjacency list which have $n - 1$ elements, hence we get complexity O(n^2).

Let us take an example (test case #5):



```
10 3
2 1 6
4 0 4 8 9
5 3 5 6 7 8
1 2
5 1 5 7 8 9
2 2 4
3 0 2 8
3 2 4 9
4 1 2 4 6
3 1 4 7
```

Figure 2: Test case 5

$n = 10$
$S = 3$
Follow the algorithm for BFS:

- $QNode = 3$ (line 1). Currently, the distance is 0 (only start node). Go through all node with this distance, $sum_0 = 3$ and $QNode = 2$ (node 2 is the only neighbor of 3).
- Similarly, visit node with distance 1 from (S). $sum_1 = 2$ and $QNode = 5, 6, 7, 8$ (node 2 have 5 neighbors 3,5,6,7,8, however, 3 is already visited).
- Visit nodes with distance 2 from (S). $sum_2 = 5 + 6 + 7 + 8 = 26$ and $QNode = 4, 0, 9, 1$.
- Lastly, visit nodes with distance 3 from (S). $sum_3 = 4 + 0 + 9 + 1 = 14$ and $QNode =$ (stop BFS)

Finally, we know that the maximum shortest distance is 3, hence, max influence is 4 (at S). Calculate the influenced sum

$$\begin{aligned} influence\_sum &= 4 * sum_0 + 3 * sum_1 + 2 * sum_2 + 1 * sum_3 \\ &= 4 * 3 + 3 * 2 + 2 * 26 + 1 * 14 \\ &= 84 \end{aligned}$$

which is the answer.