Solution for HKOI Final 2011

March 15, 2011

Contents

1 WippiLeaks

1.1 Problem description

Given the encrypted string and the k^{th} frequent character of the original string. Output the original string.

1.2 Solution

Find the k^{th} frequent character in the encrypted string. Calculate the offset and the encrypted method. Convert all characters by the offset.

1.3 Common mistakes

The problem statement might be a bit long, and this may cause many contestants miss some points. Processing strings are not as easy as one might think. Some contestants wrongly handle spaces and punctuations.

Suggestions:

- Read the sample in/out and try to get the main idea of the question. Look for evidence in the problem statement.
- To read a string with space in C, use `gets(str)`. The only thing you might need to notice is when you read the integer k and character X in the first line, you may need to `gets()` one time before getting the real text. This is to read the end-of-line character in the first line.
- Remember to put an end-of-line character after the last output line.

2 Housekeeper

2.1 Problem description

There are N rooms and M housekeepers. Each housekeeper can serve the guests in neighbor rooms. We want to assign each housekeeper to a room such that the number of guests can be guest is maximized.

2.2 Solution

To make things simple, we can first divide the rooms into groups. A group consist of connected rooms. Then we just need to process the group size.

We can keep assigning housekeepers to empty rooms that can serve two guests. If there are housekeepers remaining, we can put them into room that can serve one guest. If there are still housekeepers remaining, we can put them into room that serve no guest. At last, which is the tricky case, if there are still housekeepers remaining, we need to replace the guest rooms by those housekeepers.

2.3 Common mistakes

The mistakes of the contestants are usually missing some cases or even writing a wrong algorithm. This is due to lack of testing.

Suggestions:

- If you have some observations, use this observation to create simple test cases (readable and calculable by human).
- Try to do the problem by hand, there may be surprise of solving the problems (or at least some insights).

3 Factor Factory

3.1 Problem description

Given an integer N, find a number between 1 and N where rearranging the prime factorization is the largest.

3.2 Solution

In general, we want to produce as many digits as possible. With some observation, using prime greater than 3 is impossible. For example, we won't use a 5 because we can replace it by 2×2 which has one more digit. Also, we won't use more than one 3 because we can replace 9 by $2 \times 2 \times 2$.

Therefore, we should find x and y such that $3^x \times 2^y \leq N$, where x is 0 or 1.

3.3 Common mistakes

Over 50% of the submissions do not get full mark only because of the misread of problem statement. The problem asks you to find ALL factorization of integers from 1 to N, not just N itself. This costs a lot of points.

Suggestions:

- Read the problem statement carefully. This is hard for everyone, but the sample cases can always help you if you are in doubt. Sample 2 here is a good example. You can ALWAYS assume the example cases are correct. In some rare cases where examples are wrong, 'big cows' will ask for clarification.
- If you have no idea to the question, try to use the computer to generate answers for small cases. The most important observations can be seen here easily.
- If you have observations and have time, try to prove the observation to ensure correctness. Never sacrifice correctness for efficiency!

4 Pyramid

4.1 Problem description

Give a sequence p with length $2 \times H - 1$. Calculate the minimum number of element to be changed in p so that the final sequence has the following 2 properties.

- $p_{i+1} = p_i + 1$ for $1 \leq i \leq H - 1$
- $p_{i+1} = p_i - 1$ for $H \leq i \leq 2 \times H - 2$

4.2 Solution 1

Find another sequence b where

- $b_i = p_i - i$ for $1 \leq i \leq H - 1$
- $b_i = p_i - (i - H)$ for $H \leq i \leq 2 \times H - 2$

The most frequent number in b are corresponding element in p that should not be changed. The time complexity is depended on the approach of finding the the most frequent number. If bubble sort is applying, the complexity will be $O(H^2)$.

4.3 Solution 2

Observe that at least 1 element is not changed. We just try to fix the value of a element and then find the number of element to be changed. The complexity is also $O(H^2)$ because $2 \times H - 1$ element need to be fixed and $2 \times H - 1$ steps is need for each calculation.

4.4 Common mistakes

This is a really difficult question. Most contestants cannot find the critical observation, and some do not get full marks as they miss 'the highest pyramid' point. This problem is already simplified. The output in the original version might be more than 100,000.

Suggestions:

- Try to create test cases manually and look for observation.
- Check the constraints and create some corner cases.
- Check your program for cases of ties (as in here) to see if your program performs as expected.

5 Nuclear Reactor

5.1 Problem description

There are M weighted points on the plane $(0, 0)$ to (N, N). We need to divide the points into two parts vertically or diagonally such that the total weight of the two parts equals.

5.2 Solution 1

Try all possible placement of the separator. Then, for all points, determine it is on the left of the separator, on the right of the separator, or on the separator exactly. It runs in $O(NM)$.

5.3 Observation

Observe the case the separator is vertical. Points on (X, Y_1) or (X, Y_2) do not make a difference! That is, we can ignore the Y-coordinate if $Dir = 0$ and reduce the problem to one dimensional.

When the separator is diagonal, imagine that we project all points to X-axis as the same direction of the separator. Points at (X_1, Y_1) and (X_2, Y_2) are same if $X_1 - Y_1 = X_2 - Y_2$.

Observe that N is very small, we can process on the axis instead of processing points.

5.4 Solution 2

In the case $Dir = 0$, for each point (X_i, Y_i, S_i), add S_i to $Total[X_i]$. Similarly, in the case $Dir = 1$, we project each particle to the X-axis by the diagonal. That is, for each point (X_i, Y_i, S_i), add S_i to $Total[X_i - Y_i]$.

Then we try to put the separator which intersect X-axis at a. If $Total[a - 1] + Total[a - 2] + ...Total[-N] = Total[a + 1] + Total[a + 2] + ...Total[N]$, then a is a valid answer. Try all a from $-N$ to N.

The time complexity for this solution is $O(N^2 + M)$, which is enough to get full marks. $O(N + M)$ is also possible using some tricks.

5.5 Common mistakes

This is a bit out of our expectations. The solution seems quite straightforward, but contestants are wrong in different cases.

- Arrays are too small
- Corner cases, such as answer is N or 0
- Negative answers
- Did not consider points lying on the separator

Suggestions:
Think about the extreme test cases. Here are several easy to create examples:

- N=100, M=1, at $(0, 100)$, Dir=1

- M=5, all points on a line, Dir=0 or 1
- M=1, at $(N, 0)$, Dir=0

You can practice the above by submitting your solution to the judge only when you are very confident the solution is correct, i.e. do comprehensive testing yourself.

6 Spectrum Identification

6.1 Problem description

Given two strings S and P, both contains 'B', 'G', 'W' only, find a rotation of P such that the difference between S and $P + P + ... + P$ is smallest. The difference between two strings is defined as the number of different characters at the same position.

6.2 Solution 1

Try all M rotations of P. Then, generate the string $P + P + ... + P$ (The concatenation of many Ps) and compare with S, count the number of differences and update the best answer we have ever seen. Since we need to process the whole string of S each time, it has a complexity of $O(NM)$.

6.3 Observation

Observe that we are comparing $S[x], S[x + M], S[x + 2 \times M]...$ with $P[x]$, we can try to simplify this process.

We now want to 'compress' $S[x], S[x + M], S[x + 2 \times M]...$ to something simple. Observe that the are only 3 types of characters, we can simply store the occurrence of each type of characters.

6.4 Solution 2

First, count the occurences of each color at position $S[x \bmod M]$. For example, store the number of 'G' in $S[x], S[x + M], S[x + 2 \times M]...$ at $Grey[x]$.

Then, generate all rotations of P. Process each character $P[i]$, we can find the number of difference by adding the occurences of the other two colors. For example, if $P[i]$ is grey for some rotation of P, then the difference in this position is $Black[i] + White[i]$.

We need $O(N)$ for counting occurences and $O(M^2)$ for each rotation, the total time complexity is $O(N + M^2)$.

6.5 Common mistakes

The test cases here are less tricky than Q1. Most people are wrong due to time limit exceeded. Others are wrong on cases where $N < M$.

Suggestions: Try to generate your own large test cases by writing a simple program. You can test if your program produces very strange results/time limit exceed.

Again, create some critical test cases.

- $N < M$
- M=1
- Answer=0

7 Gravity Game

7.1 Problem description

Given a $N \times M$ map with some obstacles, simulate the movement of a ball. Each time it will move in a given direction until it hits the boundary or an obstacle.

7.2 Solution 1

Simply do what the ball does! However, it may run very slow if the movement of each command is large. For example, the largest possible movement is $Max(N, M) - 1$. In this case, the time complexity will be $O(Max(N, M) \times C)$.

7.3 Observation

Solution 1 spends most of the time on moving on free space. Hence, we try to reduce the time on moving on free space.

We can assume that the whole box is surrounded by obstacles. Therefore, there are at least 2 obstacles in each row and column. Consider two obstacles located at (X, Y_1) and (X, Y_2) $(Y_1 < Y_2)$, and there are no other obstacles between them. If the ball is located between them and started move left, it will stop at $(X, Y_1 + 1)$. Similarly, if it move right, it will stop at $(X, Y_2 - 1)$.

7.4 Solution 2

The solution is based on **precomputing** all possible movement. For example, $Right[i][j]$ denotes the final position of the ball if it starts moving right from (i, j).

For each row, process the data from left to right once. When we found an obstacle, we can process the grid between it and the previous obstacle. For example, if we found an obstacle at (3,14) and the previous obstacle is locate at (3,7), then we can fill (3,13) to $Right$[3][6] to $Right$[3][13].

Using similar method, we can also compute $Left[i][j], Up[i][j], Down[i][j]$. Then, we can immediately know the result of each command. For example, if the ball is located at (x, y) and start moving down, it will stop at $Down[x][y]$ eventually. Then we can add the distance between (x, y) and $Down[x][y]$ to our answer and update the position of the ball.

Time complexity for precomputing: $O(NM)$.
Time complexity for each command: $O(1)$.
Total time complexity: $O(NM + C)$.

7.5 Common mistakes

Reading large input must be very careful. Some contestants received Time Limit Exceed because they are reading the input slowly. Others are wrong because the simulation is wrong, e.g. not considering the border as obstacle. One last mistake is forgetting to use long long/int64.

Suggestions:

- Read large input by `scanf`, not `cin`

- For string that has no space character, always use `scanf("%s",s)` instead of reading characters one by one. For string that has space characters, use `gets(s)`. Don't use `getchar()`. Although it is faster, many unknown characters, such as end-of-line character and space, are mistakenly read
- Ensure the string array is big enough
- To make the program less error-prone, I would suggest the following segment:

```
int dx[]={1,0,-1,0},dy[]={0,1,0,-1};
int nx,ny,cx,cy; //new (x,y) and current (x,y)
for (int i=0; i<4; i++){
    nx=cx+dx[i];
    ny=cy+dy[i];
    if (nx>=0 && nx<n && ny>=0 && ny<m) //new (x,y) is inside the grid
    do something
}
```

  Please refer to the sample code after getting accepted in HKOJ.
- Estimate the upper bound of the answer.

8 Circular Circuit

8.1 Problem description

There are several points to notice:

- The component is stable depends only on its left and right components, not itself
- It is guaranteed the solution exists.

Later we will use these observations to solve the problem.

8.2 Solution 1

Note that the 50% constraint is $N \leq 10$. We can simply generate all possible permutation of the components. Then we can know if this permutation (configuration) is stable or not.

The complexity here is $O(N \times N!)$, which is enough to get 50%.

8.3 Observation

Notice that another 25% is $1 \leq V_i \leq 3$.

If $C \geq 2$, it is obvious that any configuration is stable, since the maximum difference of voltage is only 3-1=2.

If $C = 0$, we can divide it into 2 cases. First, if N is odd, every element must be the same. Second, if N is even, there are two groups of elements of equal voltage. Each group has a size of $N/2$. We can put them alternatively to create a valid configuration.

Last, if $C = 1$, notice that for a component, there are only 3 cases for the components L and R on its left and right:

- L and R are equal.
- L=1 and R=2 (or vice versa).
- L=2 and R=3 (or vice versa).

Counting the number of 1, 2 and 3 respectively, we can distribute the numbers into the circular circuit according to the above cases.

8.4 Solution 2

As we proceed, we note that in the case $C = 0$, if N is even, we can divide the components into 2 groups. Each group is independent. Hence we only need to find a stable configuration for each individual group, and the case N is odd.

Using point one in the first section, we can reduce the alternative components to a cycle, and find a configuration that two adjacent components are not greater than C.

For example: let the final configuration be ABCDE. The voltage of these pairs must be not greater than C: AC, CE, EB, BD and DA. Wow, it forms a simple cycle ACEBD. In this way we can reduce

the problem to finding a stable configuration for adjacent elements.

Using point two in the first section, we know that it is no bad to place the components from lowest voltage to highest voltage. Hence we can sort the voltage of the components first, where $V_1 \leq V_2 \leq ... \leq V_n$.

Then, consider the above example, assume we place the V_1 in A, then C can be V_2 (and this must be stable as mentioned in point two). Here comes a difficult problem, where should we put V_3? It must be either E or D. Assume we place it at E, it is not hard to create a case where V_4 and V_1 are not stable, i.e. $V_4 - V_1 > C$. That's why we must put V_3 at D. Similarly, we can put the components alternatively on each side, this will also be stable according to point two. In the example, we put the components in the order ACDEB.

The above solves N is odd. When N is even, we simply divide the components into two groups in a suitable way. The suitable way is obviously dividing the components into small and large voltage elements, which must form a stable configuration as assured in the problem.

8.5 Common mistake

This is a typical ad-hoc problem, i.e. no well known solution or common solution. This kind of problem is usually harder to prove its correctness or create test cases that can 'kill' your program. Contestants are usually wrong in indexing and processing different cases.

Suggestions:

- Debug the program by manually making small cases and output the index. Check with your hand to see if the index is correct.
- Randomly generate several test cases yourself. You can generate test cases by creating the answer first and shuffle them. Sometimes correct insights and solutions will come up in your mind when you are generating this way.
- Again, make sure your program works for corner cases (as mentioned in the solution).