



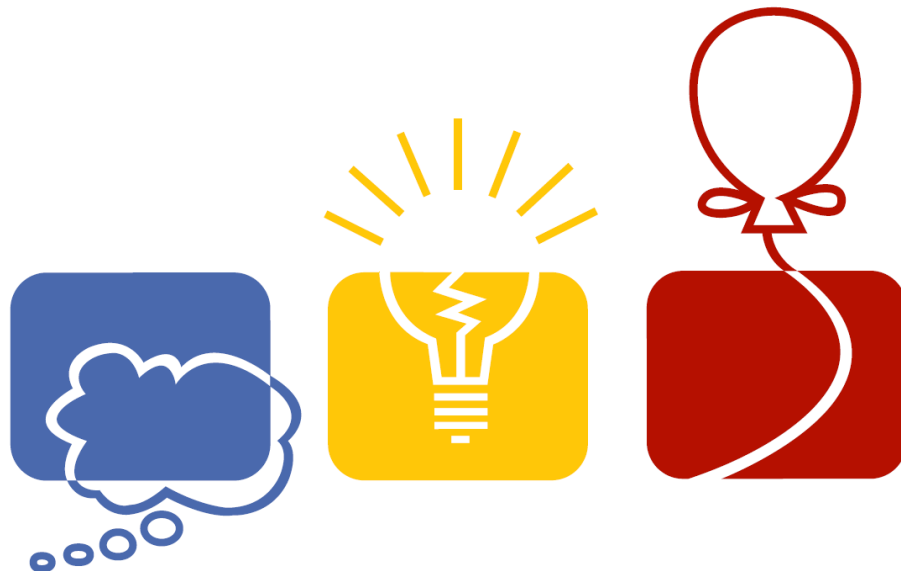
Arab and North Africa Regional Contest 2009

The Unofficial Problem Set Analysis

Written by: Khaled Hafez (khaled912@gmail.com)

Soha Sultan (sohamohsen@gmail.com)

Reviewed by: Hamza Darwish (cpphamza@yahoo.com)



This document contains detailed explanation for the solutions of the problem set of the Arab and North Africa Regional Contest 2009. You are advised to have fun trying to solve the problems before reading this document. Please note that the solutions described here are not official, and any mistakes there do not imply a mistake in the actual judge's solution.

[A] SEINFELD

There is a straight forward dynamic programming approach to this problem. The state has 2 parameters: the count of the left unmatched braces so far and the second is an index to the current brace in the given string. For every state, the optimal result is the minimum of either changing the brace orientation or leaving it as it is. The count of unmatched left braces should be incremented or decremented according to its state after the change. Pseudo-code follows:

```
function go(cnt, index)
    if s[index] == '{' then
        return min(go(cnt + 1, index + 1), 1 + go(cnt - 1, index + 1))
    else if s[index] == '}' then
        return min(go(cnt - 1, index + 1), 1 + go(cnt + 1, index + 1))
    end if
end function
```

This solution is $O(N^2)$, and given the limit for N , it should be sufficient to solve the problem.

In fact, this problem is much more interesting than what you might think. There is a $O(N)$ greedy approach that can find the minimum number of braces to flip. The idea is to scan through the string keeping track of the number of opened but unmatched braces. Whenever this count drops below zero, then the current bracket **MUST** be flipped, otherwise, the expression will never be balanced. We will keep track of how many times this happened during the scan and call this number the “overmatch” count. After we eliminate all those kind of braces, we’ll start scanning again from the beginning. At this point, we should agree that all the brace flips we made in the previous scan were unavoidable. In the second scan, we will also keep track of the number of opened but unmatched braces. This time, this number will never drop below 0. After we finish the scan, let’s say we have X opened but unmatched braces. It is obvious that the minimum number of flips needed to fix the expression is $X/2$. Also, since the input string size is guaranteed to be even, X will always be even, and $X/2$ will always be a whole number. If we start flipping closed braces from the end of the string, we are guaranteed not to create a situation where we will have an overmatched expression of braces. The final answer is the “overmatch” count + $X/2$.

[B] TILES OF TETRIS, NOT!

The actual given dimensions of the rectangle do not matter; what matters is the actual ratio between them. To calculate the ratio, we divide both dimensions by their greatest common divisor (GCD). Call the reduced dimensions (X, Y) . Now we are looking for the shortest side length that is divisible by both X and Y , which is the least common multiple of X and Y . Since X and Y are relatively prime, this number is simply $S = X*Y$. The number of horizontal rectangles that would form the desired square is S/X , while the number of vertical ones is S/Y . The total number of needed rectangles is $(S/X)*(S/Y)$. However, we just said that $S = X*Y$. By substitution we get that the total number of rectangles as $(X*Y/X)*(X*Y/Y) = X*Y$. A common mistake was to use 32-bit integers when doing the multiplications causing an overflow.

[C] NOT SO FLAT AFTER ALL

Although the description of this problem is a bit complicated, the problem is only about prime factorization. The distance between the two integers in the N -dimensional space is the difference of the powers of the union set of the prime factors of the two integers. We could store a sorted set of prime factors with their corresponding set of powers for each of the given integers. The required distance is the sum of the absolute difference of powers between the corresponding prime factors. For the prime factors that exist in one of the given integer but not the other, you can assume that the power of that prime in the other integer is 0.

[D] PROBABILITY ONE

Not much to say about this problem. Just follow the steps mentioned in the problem statement. One of the teams managed to solve it in the first 4 minutes during the actual contest.

[E] HOP – DON'T WALK!

The problem asks to transform the board into the following: a sequence of Ws followed by a sequence of Bs followed by a sequence of Ws. The frog could stand anywhere. We will refer to all those states as 'final' states. There are only 4 types of movements and we have a very nice constraint that if the number of required movements exceeds 9, we print -1. This gives room for brute force: we can try out all possible sequence of movements and see the one that would lead to a final state. The most straight forward brute force leads to $4^9=262,144$ possible sequences to try. One way to try all possible sequences is through recursive backtracking. A faster method (time-needed-to-code-wise) is to do an iterative loop over a bitmask from 0 to $2^{18}-1$, and take the integer representing each two bits as an ID to the movement. Let's say "Move Left" is 0, "Move Right" is 1, "Hop Left" is 2 and "Hop Right" is 3. Here is an example of how to interpret a bitmask as a sequence of movements:

Bitmask	01	01	10	00	11	10	11	10	00
Corresponding Ids	1	1	2	0	3	2	3	2	0

Corresponding sequence: Move Right, Move Right, Hop Left, Move Left, Hop Right, Hop Left, Hop Right, Hop Left, Move Left.

Although the solution described above is sufficient to solve the problem, there is a neat trick that could bring down the complexity to $N*3^k$ instead of $N*4^k$ (where K is the maximum number of allowed movements). The thing to notice is that it is meaningless to make a certain movement at turn X and then do the exact opposite movement at turn X + 1. For example, "Hop Left" would not be followed by "Hop Right", "Move Left" would not be followed by "Move Right", and vice versa for both cases. So now, at each turn, we only have 3 choices instead of 4, reducing the complexity to $N*3^k$.

[F] AIR STRIKE

For the first while, the search space for this problem seems infinite. Some people might think that since we have limited precision, we could loop over “all” possible values for the power distributions. However, from the given limits, those solutions should exceed any reasonable time limit. A little trick can limit our search space only to $2n$ candidate energy distributions. The trick is to observe that given any solution that deflects a certain number of missiles, we can construct another solution that deflects the same number of missiles where one of the missiles is touching the border of one of the disks. As defined in the problem statement, the optimal solution is the one that maximizes the total number of missiles deflected by the two disks. Suppose we have an optimal solution where no missiles hit the boundaries of the disks. Take any of the disks and start increasing the energy allocated to it (and remember this automatically decreases the energy allocated to the other disk) in this optimal distribution until:

- 1- The disk whose area is being increased is about to include a missile's landing position. If this increase in area does not cause the other disk to exclude any missile positions at the exact instant, then the original configuration was not optimal and this contradicts the assumption. Otherwise we still have the exact same number of missiles we started with before the inclusion/exclusion, so we can continue giving more energy.
- 2- The other disk (whose area is decreasing) is about to touch a missile's landing position. This means that any further increase in the area for the first disk/decrease in the area for the second disk will cause the second disk to exclude a missile position and the solution will not become optimal.

According to the above observation, every missile defines two candidate optimal solutions. For every candidate, the number of missiles being deflected by either of the two towers can be easily calculated by testing how many missile landing locations lie inside the disk of either towers. This leaves us with a $O(n^2)$ solution. Here is a quick exercise for you: can we decrease the number of candidate optimal power distributions to n instead of $2n$?

[G] STOCK

This problem does not involve any complex algorithms. The key idea is to keep your graph in the transitive closure form all the time (i.e. $a \rightarrow b$ and $b \rightarrow c$ implies $a \rightarrow c$, where \rightarrow represents a directed edge). If you think (just think, not use) Floyd Warshall, you would know that you can achieve fully connectivity in the graph after a maximum of $O(N^3)$ iterations. If your graph is fully connected, you will be able to answer each query in $O(1)$. Pseudo-code for the core function that adds an edge and keeps the graph in the transitive closure is as follows:

```
function add(source_node, target_node)
    if source_node is connected to target_node then return
    connect source_node to target_node
    for each node i that has an edge incident from target_node
        add(source_node, i)
    end for
    for each node j that has an edge incident to source_node
        add(j, target_node)
    end for
end function
```

Although the above function looks simple, its analysis is not as easy. Let's start by the complexity analysis: for each test case, there are N possible values for `source_node` and N possible values for `target_node`. The results are cached; therefore, each unique pair of (`source_node`, `target_node`) will be computed exactly once. Each one of the computations involve two linear loops, for an overall complexity of $O(N^3+T)$ per test case. This is not easy to see immediately, but if you think about it and trace the pseudo code for a while with a pencil and a paper, it should become clear. One more thing that could cross your mind is that caching intermediate results could lead to an incorrect solution. This is not true because of our initial assumption: the graph is always kept in the transitive closure form. The presence of an edge from X to Y implies that X is already connected to all nodes that Y is connected to. Here are few exercises that should help you have a more clear understanding of the solution (you are advised to solve them in the given order):

- What is the maximum recursion depth of the function *add*?
- If we change *add* to the following iterative version, will we still be able to achieve a $O(N^3+T)$ time complexity?

```
function add(source_node, target_node)
    if source_node is connected to target_node then return false
    connect source_node to target_node
    for each node i that has an edge incident from target_node
        for each node j that has an edge incident to source_node
            connect i to j
        end for
    end for
    return true
end function
```

[H] LAND DIVISION

The first thing to notice is that this problem can be converted into two 1-dimensional sub-problems by projecting the points one time on the X-Axis and another time on the Y-Axis. Now we're trying to solve the following: given a sequence of city locations, we want to partition them into $K+1$ subsequences minimizing the average of the difference between each group size and N/K . The second thing to notice is that minimizing the average is equivalent to minimizing the sum. Let's start by establishing a naïve $O(N^2K)$ solution and see what we can do about it: The state is (current_city, remaining_sons). In each state, we try to assign all possible subsequences (parts) of cities starting at current_city to one of the sons, and take the decision to take the group that minimizes the total. The key to the solution is to notice that we don't need to try all possible sizes of a certain part. If we can do a perfect division at a certain state (current_city, remaining_sons) where the size of the part could be equal to the baseline, then it is always optimal to make this division. If it is not possible to do a perfect division, then if you think about it, the optimal solution will involve having the size of the part equal to either the first possible size less than the baseline or the first possible size greater than the baseline. To quickly find those locations, binary search could come in handy. The time complexity of this solution is $O(K*N*\log N)$ and the memory complexity is $O(K*N)$.

Another approach is brute force based on the same observation: in the optimal division, each son will take a number of cities that is either just above the baseline or just below the baseline. Since we don't know which son will have a number of cities just above the baseline and which will have a number of cities just below the baseline, we will try out all possible 2^k configurations. For each configuration, we do K binary searches to assign the appropriate amount to each son, for an overall time complexity of $O(2^k * K * \log N)$ and a memory complexity of $O(N)$.

Important note: although the required result is a fraction, it is possible to do all the calculations without implementing a "fraction class". The optimal solution is always a multiple of K^2 . Just multiply all relevant given integers by K^2 , compute the integer value, and then divide and reduce before printing the result.

[I] KIND OF A BLUR

Each given pixel is the average of few others that surround it. Let's take a general example of a 2x2 image with a blurring distance of 1.

x y	a b
z w	c d
Original Image	Blurred Version

We are given a, b, c, and d and we need to figure out x, y, z and w. One thing we know is that $a = (x+y+z)/3$. The key to the solution is to rewrite this as: $x + y + z = 3a$. Same for b, c, and d:

$$x + y + w = 3b$$

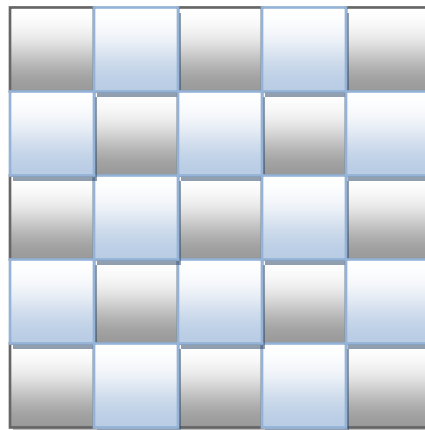
$$w + z + x = 3c$$

$$y + w + z = 3d$$

How can we figure out x, y, z, and w? The way we rewrote the equations now makes it very simple: we need to solve 4 equations in 4 unknowns. If you now think about the general $N \times N$ case, you will have $N \times N$ equations in $N \times N$ unknowns. The blurring distance parameter will only affect the coefficients of the equations. As the limit for N is 10, you will have a maximum of 100 equations in 100 unknowns, which can be solved directly in $O(M^3)$ using Gauss Jordan elimination (where M is the number of equations) for an overall complexity of $O(N^6)$.

[J] NATIONAL TREASURE

This problem can be reduced to finding the maximum independent subset in a graph. The graph can be constructed as follows: Each artifact in the museum represent a node. There is an edge between the nodes representing two artifacts if they conflict – i.e: they cannot be kept simultaneously in the room. This is when the critical points come into play. If artifact X has a critical point at artifact Y's location, then keeping X implies that we cannot keep Y. It could be confusing for the first while as you might think that edges are supposed to be directed, but think about it: if keeping X implies we cannot keep Y then keeping Y implies we cannot keep X, even if X is not located at one of Y's critical points. Now we have our graph. But wait a minute, isn't finding the maximum independent subset in a general graph an NP-Complete problem? This is true, but there is one case where this problem is solvable in polynomial time: if the graph is bipartite. So now we need to see if the constructed graph is bipartite. In fact, it is. We will show that through showing the partition of the graph. In fact, a “checkerboard” partition could do the trick (light cells represent the first part while dark cells represent the second part).



If we say that an artifact is placed on one of the above cells, you can never have a critical point for this artifact in a cell that has the same color. That means that there can be no edges between the nodes representing artifacts placed on cells of the same color. From here, the rest of the solution is easy to construct. We are asked to minimize the number of artifacts to eliminate all conflicts. This is equivalent to maximizing the number of artifacts to keep. No pair of nodes from those representing the artifacts-to-keep should have an edge between them, which exactly describes the maximum independent subset. In a bipartite graph, this is equal to $N - M$, where N is the number of nodes in the graph and M is the size of the maximum matching. You are advised to search the web for König's theorem to get a clear view of how this relation holds. You might encounter a different term, which is the “minimum vertex cover”. This is the exact opposite of the maximum independent subset. Any graph is partitioned into a minimum vertex cover and a maximum independent subset.