CROATIAN OPEN COMPETITION IN INFORMATICS

2nd ROUND

SOLUTIONS

COCI 2009/2010 **Task FAKTOR**

2nd round, 21. November 2009.

Author: Marko Ivanković, Goran Žužić

The formula is simply a * b – a + 1.
Alternatively one can solve this task by brute force search. The solution is a number between 1 and 100*100 so you can easily try all possible answers and choose the smallest one that fits.

Necesarry skills:
mathematics

Tags:
ad-hoc

COCI 2009/2010 **Task RIMSKI**

2nd round, 21. November 2009. **Author:** Filip Barl

Let us examine all possible cases that require modification of the input sequence. We can easily see that such cases will end up producing 4 or 9 on their ones place, and 40 on the tens place. All numbers ending with 6 (6 itself, 16, 26, 36, 46, etc) can be transformed into corresponding number ending with 4 such as 6 (VI) to 4 (IV), 16 (XVI) to 14 (XIV) etc. You might have noticed at this point that one number, 66, will not transform in 64. Will will account for that later on. Also, some, but not all, numbers ending with 1 can be transformed to end with 9. For example 11 (XI) transforms into 9 (IX), 21 (XXI) into 19 (XIX), 31 (XXXI) into 29 (XXIX), 41 (XLI) does NOT transform into 39 however, neither does 51 (LI) into 49 (IL). The correct way to write 49 would be (XLIX). 61 (LXI) can be transformed into 59 (LIX) so can 71 into 69 and 81 into 71. 91 (CXI) cannot transform into 89 (LXXXIX). Now we have covered all cases that require displacement of ones. We just need to account for numbers between 60 (LX) and 69 (LXIX). These numbers can be transformed into 40 – 49 range by displacing the first X before L. Note that we will now cover the special case of 66 and 71. 66 (LXVI) can be first transformed into 46 (XLVI) and than into 44 (XLIV). 71 (LXXI) can transform into 69 (LXIX) but 69 can transform into 49 (XLIX).

Of course instead of going through all possible special cases, you can count all available characters and construct the smallest possible number. Which is what we did.

Necesarry skills:

listing all possible cases

Tags:

ad-hoc

COCI 2009/2010 **Task KUTEVI**

2nd round, 21. November 2009. **Author:** Bruno Rahle

Let Ak be a set of angles {a1, a2, a3, …. an } that Mirko can construct using at most k additions or subtractions of the initial angles. From Ak we obtain A(k+1) by copying Ak into A(k + 1) and then for each initial angle adding it to all angles in Ak inserting the resulting angle in A(k + 1), if it is not already in. It can be easily seen that Ak can contain at most 360 angles, because there are only 360 possible values. Because the smallest number of angles we can add to A(k + 1) is 1, because if we did not add any new angles we could terminate our algorithm, we will perform at most 360 steps at which point A(360) will contain all angles Mirko can construct. Of course A(0) = {0}.
This can easily translate into a efficient algorithm for this task. First we precalculate A(360) and than for each angle Slavko gives, simply check if it is contained in A(360).

Necesarry skills:
modular arithmetic

Tags:
dynamic programming

COCI 2009/2010 **Task VUK**

2nd round, 21. November 2009. **Author:** Filip Barl

First, let us find for each patch what is the smallest distance to some tree. We do this by running a BFS algorithm, we start at all patches containing a tree, marking them with distance 0. Afterwards we expand in a BFS manner increasing the distance by 1 for each patch. This can be done in O(N * M). Now let us search for the best path from Vjekoslavs starting position. We maintain a sorted data structure (a heap, for example) that stores all patches not yet visited that are next to some patch we already visited. We sort the patches by decreasing tree distance. At first, the heap contains exactly one patch, Vjekoslavs starting patch. As we visit each patch, we add all his neighbors to the heap and mark it as solved. The solution for that patch is his own distance, or the solution for the patch we visited it from, whichever is smaller. We can terminate the search when we visit the cottage patch. The worst case is O(N * M * log (N * M)).

Necesarry skills:

simple graph traversals, sorted data structures

Tags:

graph algorithms

COCI 2009/2010 **Task POSLOZI**

2nd round, 21. November 2009. **Author:** Marko Ivanković

This task was a special one. It is designed just for the purpose of intriguing people to read this solution. If you are familiar with A* and Bidirectional Search, you can skip this part, if you are not, than this task was designed to get you familiar with them.

The solution to this task can be a simple state search. Let one ordering of numbers represent a state. We can search the shortest path from the starting state to the finish state using any search algorithm. BFS and DFS however, were not fast enough or were memory expensive.

One possible solution was Bidirectional BFS. Bidirectional BFS is a special type of search. It can be describe as two BFS searches, one from the starting, one from the finishing state. The advantage of bidirectional BFS is smaller time complexity compared to normal BFS. While normal BFS is O(b^d) bidirectional BFS is only O(b^(d/2)) where b is the branching factor and d is the distance from start to finish. The problem with bidirectional BFS is that the search starting from the finishing state needs to perform backward movements. In some problems this can be quite hard to calculate. In this problem, however, the backward movements were identical to the forward moves.

The other solution was A*. A* is a heuristic search that takes into account not only the distance of a state from start but the expected distance to the finish. The most important part of A* is how we calculate this expected distance. The function that calculates this expected distance (called the heuristic function) must be optimistic, and monotone. Optimistic means that for any state X, the function H(x) must be smaller than or equal to the true path from X to the finish state. Monotone means that for states X and Y where dist(X, Y) is the distance from X to Y H(X) must be smaller than or equal to H(Y) + dist(X, Y). If the heuristic is not optimistic, A* will return wrong answer. It will overestimate the shortest path and miss it. If it is not monotone, A* needs to be able to revisit nodes.

Necesarry skills:

advanced state search algorithms

Tags:
state search algorithms

COCI 2009/2010 **Task PASIJANS**

2nd round, 21. November 2009. **Author:** Goran Žužić, Luka Kalinovčić

Let us try to form a greedy strategy. The question we need to answer is, what deck do we take the card to form the best solution sequence? Let's us examine all top cards on all decks. If there is only one smallest card, it is obvious we need to choose that deck. Any other deck yields worse solution sequences. The problem is if more than one deck share the smallest card. We now examine prefixes of those decks. For simplicity we only present a case with two decks. Extension to more decks is simple. Let's examine decks A and B, both share the same prefix of length L. Let the L + 1 card in deck A have smaller value than the L + 1 card in deck B. It is preferential for us to choose deck A over deck B because when at some point we remove those L cards, if we chose deck B we get worse results than if we chose deck A. Because those L cards are equal, it is irrelevant what deck we choose up to that point. If one of the decks is a full prefix of the other, we choose the one with larger length. This gives us a greedy strategy to work with. We just need a data structure that can quickly maintain this greedy ordering. Let us examine all substrings of all decks of length 2d. There are at most 1000N of these. We sort these and use this info for fast comparison. Suppose we sorted all substring of length 2d-1. It is clear that a sequence of length 2d is formed by connecting two sequences of length 2d-1. We now have a simple way to check for ordering of two strings of length 2d. We compare the first halves, and if needed second halves. The complexity using such comparison of the greedy strategy is O(n lg2 n).

Necesarry skills:

advanced greedy strategy generation

Tags:

greedy ,data structures