SRM 600

Added by [[rng_58]] , last edited by vexorian on Dec 24, 2013 (view change)
Labels: (None) EDIT

Single Round Match 600
Saturday, December 14th, 2013

Archive
Match Overview
Discuss this match

Match summary

Would you believe it? It is SRM # 600. The nostalgic in this SRM writer has been mislead to believe that SRM 500 happened just recently, but apparently it has been 2.5 years. Time flies when you are writing editorials. This important milestone was celebrated in full. t-shirts were offered. Cash prizes were offered. Topcoders from all the world wanted a t-shirt and put their full effort.

Getting the t-shirt was not going to be easy. The problem set started with an approachable 250 by **gojira_tc** that was not without corner cases. **Petr** set the fastest score in this one. The medium problem by **ir5** was a complex brute-force / dynamic programming hybrid that required wit on top of implementation skill. **wata** was somehow able to do all that in mere 18 minutes. The mathematical skills required for the hard problem (also by **ir5**) did not stop **niyaznigmatul** from solving it in half an hour.

In order to win the division, more than speed in a single problem was needed. The first place would go to **yeputons**, due to amazingly consistent speed in all three problems and a successful challenge; The only coder beat the 1100 points mark. Also impressive: **misl4av** scored almost 900 points and got the second place. The close third and fourth places went to **niyaznigmatul** and **wata**. Division 2 experienced a tough competition for the fewer t-shirts, solving modified versions of the division 1 problems. **Alex_2oo8** got the first place with over 1400 points.

The Problems

TheShuttles | ORSolitaireDiv2 | PalindromeMatrixDiv2 | ORSolitaire | PalindromeMatrix | LotsOfLines

PalindromeMatrixDiv2

Rate It Discuss it

Used as: Division Two - Level Three:

Value	1000
Submission Rate	24 / 1290 (1.86%)
Success Rate	10 / 24 (41.67%)
High Score	**tmbao** for 725.82 points (19 mins 2 secs)
Average Score	527.10 (for 10 correct submissions)

Care only about the palindrome rows and columns

A key aspect of this problem is that it requires *at least* $r = \text{rowCount}$ rows and $c = \text{columnCount}$ columns to be palindrome. The *at least* part simplifies our search, because if doesn't matter if the remaining rows/columns are palindrome or not - We can just pick the special r rows and c columns and the condition can ignore the rest of the cells.

There are $\binom{n}{r}$ ways to pick the r rows and $\binom{m}{c}$ ways to pick the c columns (Where $\binom{n}{k}$ is the binomial coefficient, n is the total number of rows and m the total number of columns). With $n, m \leq 8$, the worst number $\binom{m}{c} = \binom{8}{4} = 70$, so there are at worst 70 different ways to pick the palindrome rows. There are also at worst 70 different ways to pick the palindrome columns. A combined choice of palindrome rows and columns would have at worst 70*70 = 4900 options. Let's iterate through all these options and find, for each of them, the minimum cost to make those columns and rows palindrome. Then just show the minimum result out of every possible pick.

> To search for all the combinations of palindrome columns and rows you can use backtracking. You can also use bitmasks to iterate through all the subsets of columns/rows and caring only about the ones with exactly r and c columns. Another option would be to use the language's libraries if available, c++ has next_permutation which could be used on a list of r elements equal to 1 and $n - r$ elements equal to 0; Python has itertools.combinations.

Minimum cost for a specific case

After we decide the r rows and c columns to be palindrome, we need the minimum cost to do so.

In one dimension

Let's first understand how the palindrome condition works in a single dimension. For example, what's the minimum cost to turn "10010101" to palindrome? The resulting string needs the following conditions to be true:

- $B_0 = B_7$
- $B_1 = B_6$
- $B_2 = B_5$
- $B_3 = B_4$

For each of the pairs $(i, n-i-1)$, you find that $B_i = B_{n-i-1}$, which means that you need the minimum cost to turn A_i and A_{n-i-1} equal. In this specific case, the cost is 2, because $A_2 \neq A_5$ and $A_3 \neq A_4$.

Two dimensions

In two dimensions, the logic ought to be more complicated. Let's try an example:

```
1010
0101
0101
1010
```


Assume that we decided that column 2 and row 2 must be palindrome. The following conditions need to apply in the final matrix for row 2:

- $B_{20} = B_{23}$
- $B_{21} = B_{22}$

For column 2:

- $B_{02} = B_{32}$
- $B_{12} = B_{22}$

If $B_{12} = B_{22}$ and $B_{21} = B_{22}$ then we can conclude that also $B_{12} = B_{21}$.

There are various groups of cells in which all cells in the group must be equal. We can find the maximal groups possible. First solve the cells involved in: $B_{12} = B_{22} = B_{21}$. The three cells must be equal. We currently have: $A_{12} = 1, A_{22} = 0, A_{21} = 1$. The minimum cost to turn all these cells equal is 1 (Change 0 to 1). In general, the minimum cost will be equal to the minimum between the number of cells with 0 and the number of cells with 1 (So you change this minimum group to the other value).

For the remaining groups, we have simple conditions like: $B_{20} = B_{23}$. Do the same, initially $A_{20} = 0$ and $A_{23} = 1$, so the cost is 1. There are cells that don't participate in any condition, but we can assume them to belong to groups of exactly one element. Like $A33$ must be equal to itself. Currently $A_{33} = 0$ so there are no '1's in this group and the minimum cost is 0.

In general

In general, it is all about finding these groups of cell that must contain only equal numbers. The useful reduction is to consider the cells as vertices and the = relationship as edges in a graph. This way the groups of cells that must be equal are connected components. We find each connected component in the graph, we can use a Depth-first-search (DFS) to visit all the nodes connected to each node we find. Let's define the edges:

- Cell (i, j) is connected to cell $(n - i - 1, j)$ if column j is palindrome ($B_i j = B_{n-i-1} j$).
- Cell (i, j) is connected to cell $(i, m - j - 1)$ if row i is palindrome.

The DFS needs $O(|V| + |E|)$ time where $|V|$ is the number of vertices and $|E|$ the number of edges. In this case, there are $O(nm)$ vertices and each of them has at most 2 edges, so the DFS will be $O(nm)$. We repeat this DFS at most 4900 times.

Code

```
int minChange(vector<string> A, int rowCount, int columnCount)
{
    int n = A.size(), m = A[0].size();
    int pcols[m];
    int prows[n];
    int res = n * m;
    // for use with next_permutation, pcols is initially: {0,0,...,0, 1,1,...1 }
    for (int i = 0; i < m; i++) {
        pcols[i] = (m - i - 1 < columnCount);
    }
    do {
        // for use with *nested* next_permutation,
        // prows is initially: {0,0,...,0, 1,1,...1 }
        for (int j = 0; j < n; j++) {
            prows[j] = (n - j - 1 < rowCount);
        }
        do {
            // did you know? Only 4900 ways to do this.
            int cost = 0;
            bool visited[8][8];
            memset( visited, false, sizeof(visited) );
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < m; j++) {
                    if (! visited[i][j] ) {
                        int o = 0, z = 0;
                        // A iterative version of DFS:
                        // count the number o of '1's and number z of '0's
                        //  in the connected component that contains (i,j)
                        stack<int> S;
                        S.push(i); S.push(j);
                        while (! S.empty() ) {
                            // (x,y) is connected to (i,j):
                            int y = S.top(); S.pop();
                            int x = S.top(); S.pop();
                            if (! visited[x][y]) {
```

At most 4 connected cells

Perhaps the DFS only complicates things. Something to notice about the graph is that a connected component has at most 4 cells. The cells are: (i,j), $(n-i-1,j)$, $(n-i-1,m-j-1)$ and $(i,m-j-1)$. For each cell (i,j) with $\left(i < \frac{n}{2}\right)$ and $\left(j < \frac{m}{2}\right)$, we can calculate the minimum cost using the four cells that were mentioned and the knowledge of whether or not rows i and $n-i-1$ are palindrome and columns j and $m-j-1$ are palindrome. Depending on the palindrome decision, two cells in the same row or column have to be equal. If at most one of the rows/columns is not palindrome, all of the four cells have to be equal. Else there are 4 cases in which 3 cells are equal and the rest are cases in which you can treat the rows and columns individually. You can find example code for this idea in the explanation for the division 1 version of the problem.

Alternative solutions and additional comments.

<Place your comments here>

Next problem: ORSolitaire

Author

By **vexorian**

TopCoder Member

Editorial feedback	Choose
I liked it.	
I didn't like it.	

Comments (Hide Comments)

"int pcols[m];"

dude, do not write like that - if 'm' is not a constant it is forbidden in C++ ! (however some compilers allow to do this...). The correct syntax is "int *pcols = new int[m];"

Posted by lukasz16a at Dec 17, 2013 12:25 | Reply To This

TopCoder's default compiler allows it, and the code is much cleaner without using pointers.

Posted by vexorian at Dec 17, 2013 13:31 | Reply To This

It is a very bad practice ; if you don't want to use pointers directly, use vector<int> instead

Posted by lukasz16a at Dec 17, 2013 17:36 | Reply To This

I'd avoid unstandardized features when they would hinder portability or when they'd become deprecated. In the case of initializing 1d arrays with variable sizes, it will be added to the c++14 standard anyway. Since the feature helps the code and will be in the standard anyway, I think it is really a matter of opinion whether you start to use it now or not.

Posted by vexorian at Dec 17, 2013 19:19Updated by **vexorian** | Reply To This

As it's in the Wiki, there's a possibility to improve it. It can be language correction, wording improvement or additional explanation in some parts, your additional comments, description of alternative solutions, etc. If you want to improve the wording of editorial writer or correct some language error, please feel free to put your change over the original text. And if you wish to add a comment or describe another approach, there's a section for this at the bottom of each problem.

Before editing, please be sure to check the guidelines.

You can also add a comment in this comment section. When posting a comment thread, make sure to specify the problem you are talking about.

Posted by vexorian at Dec 18, 2013 19:18 | Reply To This

34 secs for div2 250 by a.taheri and 21 secs for div2 500 again by a.taheri even solving skill is appreciated that is suspicious for a multiple account ... ahh people these days !!!! am I the only on who is seeing it

Posted by _maverick at Dec 21, 2013 02:57 | Reply To This

vexorian, in the explanation of DIV1 hard LotsOfLines, I think all the conditions "q < a" should be "q <= a", is it?

Posted by jfguo at Dec 24, 2013 02:09 | Reply To This

Yes.

Posted by vexorian at Dec 24, 2013 06:54 | Reply To This

How is the value of "c" 4, in
http://apps.topcoder.com/wiki/display/tc/SRM+600?showComments=true&editComment=true&focusedCommentId=139199378#PalindromeMatrixDiv2?

Posted by innocentboy3 at Feb 22, 2014 15:14Updated by **innocentboy3** | Reply To This

Add Comment