TopCoder Competitions

SRM 608

Added by vexorian , last edited by vexorian on Feb 14, 2014 (view change)
Labels: (None) EDIT

Single Round Match 608
Friday, February 7th, 2014

Match summary

This problem set was brought to you by algorithm director **rng_58**. A tough match that put emphasis in mathematical thought. Division 1 coders began having to solve a problem that required some good analysis to reveal the greedy solution. **Petr** solved this problem in seemingly supernatural 4 minutes. The mathematical reasoning was not missing from the graph theory division 1 medium either, **Petr** also had a stunning speed in this problem solving it correctly in under 12 minutes. Many great coders solved the division 1 hard, but **WJMZBMR** did it in glorious 17 minutes. Some similar speed records (second fastest submission in medium, fourth fastest submission in easy), allowed **WJMZBMR** to score a significant division win over **Petr** (2nd place), **ainu7** (3rd), **tomek** (4th) and **mR.ilchii** (5th). Division 2 actually faced similar themes and complexities, with slightly simpler versions of the division 1 easy and medium. The fastest submission in their division 2 problem gave **andyooo** the division 2 win.

The Problems

OneDimensionalRobotEasy | MysticAndCandiesEasy | BigOEasy | MysticAndCandies | BigO | OneDimensionalRobot

BigOEasy

Used as: Division Two - Level Three:

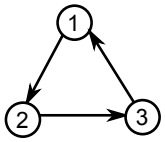
Value	1000
Submission Rate	106 / 1199 (8.84%)
Success Rate	70 / 106 (66.04%)
High Score	**andyooo** for 803.79 points (14 mins 48 secs)
Average Score	514.77 (for 70 correct submissions)

An unbounded graph contains cycles

Imagine a graph without cycles, a Directed Acyclic Graph (DAG). The number of walks in this graph will be finite. Just count the number of paths in the graph. (The difference between walk and path is that walks can visit the same vertex multiple times, in a DAG, this is not possible). If the number of walks is finite, then it is already bounded by a polynomial - a degree zero polynomial. Take $P(L) = L^0 + W$, where W is the finite number of walks in the graph for any L, $P(L) \geq L$.

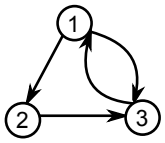
If there are cycles

If there are cycles, the number of walks of length L may still be bounded by a polynomial. A simple example:



There are 3 walks of length 1: 1, 2, 3. Three walks of length 2: 1->2, 2->3, 3>1. In fact, for every length $L > 0$, there are always 3 walks. The number of walks is infinite, but the number of walks of each length is constant.

An unbounded case would be something like this:

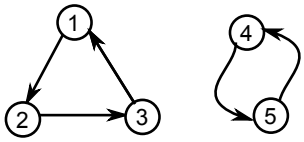


There are two ways ways to move from vertex 1 to vertex 3 and back. One needs 2 steps: (1 -> 3 -> 1), the other 3 steps: (1 -> 2 -> 3 -> 1). This means there are at least 2 different cyclic walks of length 6, that start and finish at vertex 1:

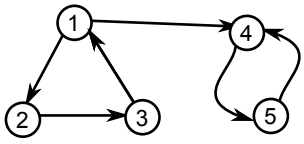
1 -> 3 -> 1 -> 3 -> 1 -> 3 -> 1
1 -> 2 -> 3 -> 1 -> 2 -> 3 -> 1

For each walk of length $L - 6$ that includes vertex 1, we can create 2 different ways of length L. (Add a cycle starting at 1 and finishing at 1) Which also means there 4 different ways to make a walk of length $L + 6$, 8 different ways for length $L + 12$ and so and so. This shows that the number of walks grows at least exponentially.

This doesn't mean that having two or more distinct cycles in the graph makes it necessarily unbounded. A trivial counter-example:



So there are always 5 walks of each length (3 + 2). But even when the cycles are connected, the graph is still not necessarily unbounded:



There is always a walk of each length that begins at node 4: 4, 4 -> 5, 4 -> 5 -> 4, etc. For each walk of length L that begins at 4, we can create a walk of length $L + 1$ that begins at node 1. (Just start at node 1 and move to 4 and do the walk). Because of the cycle starting at 1, for each of those walks of length $L + 1$, we can create new walks of lengths: $L + 3$, $L + 6$, $L + 9$, etc; by repeating the cycle that starts at node 1. Now imagine we want to know the number of walks of length L, for each walk of length $L - 3x$, there is a walk of length L. This hints us that the number of walks is linear - a polynomial of degree 1.

The difference between this case and the exponential case that had two cycles is simply that the two cycles overlap in the exponential case. Try to prove this, if there are overlapping cycles, the number of walks grows exponentially. If there are no cycles or the cycles don't overlap, the number of walks is bounded by a polynomial.

Overlapping cycles

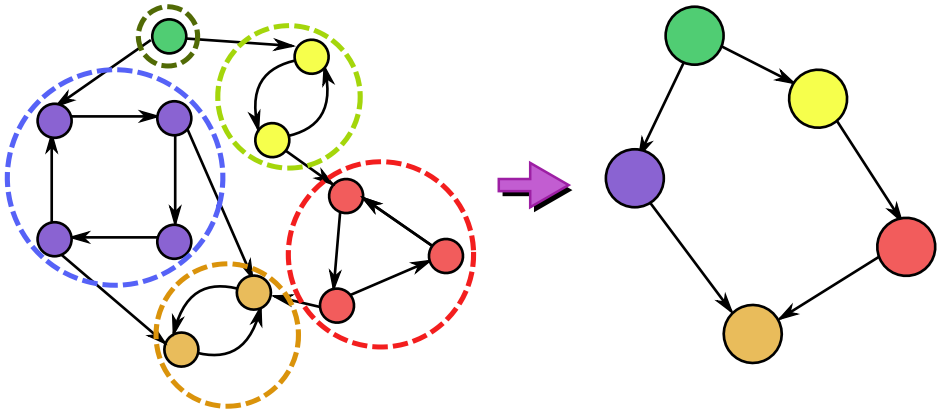
If a graph has overlapping cycles, the number of walks grows exponentially. The proof is a general version of what we did with the case above. Assume there are two distinct cycles that start at a node i. The cycles have lengths u and v. This means that there is a cyclic walk of length uv that starts at node i. For each walk of length L that contains node i, there are at least two distinct walks of length $L + uv$. Which means that there are four distinct walks of length $L + 2uv$ and so and so.

No overlapping cycles

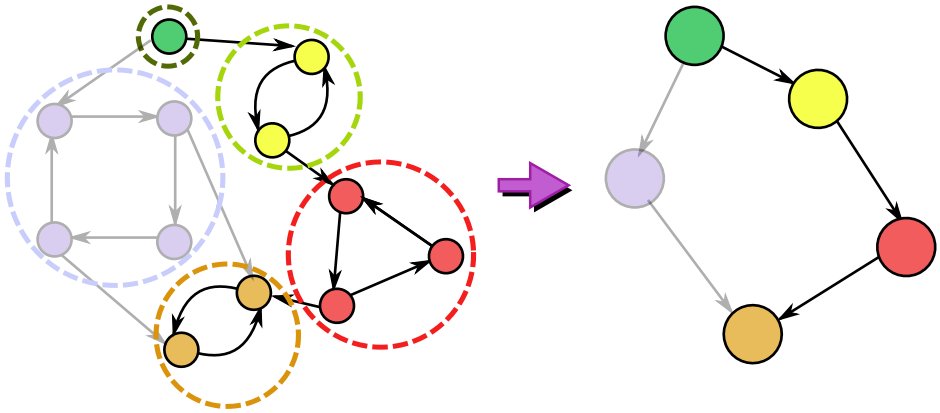
Does a polynomial bound the number of walks in any graph with no overlapping cycles?

Strongly-connected components

Something useful is to find the strongly-connected components of the graph. When there are no overlapping cycles, each strongly connected component will be a single vertex or a single cycle. After finding the SCCs, we can make a DAG where nodes represent each SCC.



We can easily find walks in the new graph and we can find a relationship between the walks in this graph and the walks in the original one. For each walk in the new graph, we can count the number of different walks that visit the SCCs in the original walk. The walk with the worst complexity will determine the complexity of the total graph. So let's focus on each of these walks and analyze its complexity:



In the new graph, each walk is a non-cyclic sequence of SCCs. Let's calculate the complexity of a single walk. Begin with the last SCC in the walk. This SCC is just a single cycle. We already know that the number of walks in a single cycle is constant.

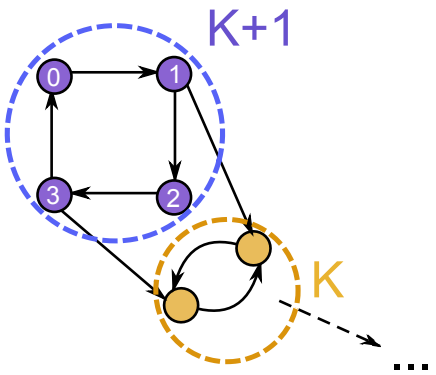
Then we move to the second-last SCC in the walk. This is another cycle. Note that the last two SCCs in this walk make a figure similar to the fourth image in this explanation. We already found that the result in this case was a linear polynomial.

Starting on the third-last SCC, you shall find the polynomial will have degree 2.

We need a formal way to conclude that the complexity will be polynomial. Through induction: We know that a sequence of just 1 SCC can be bounded by a polynomial. Now we should prove that, if it is true for a sequence of K SCCs, it should be true for a sequence of $K+1$ SCCs. So we have K SCCs in a sequence and we know that there is a polynomial $P(L)$ such that, for each L, the number of walks in these K SCCs is less than or equal to $P(L)$.

The (K+1)-th SCC will either be a single vertex or a cycle. If it is a vertex then there is a single edge connecting the (K+1)-th SCC with the K-th SCC. Each walk of length $L-1$ starting at the connected vertex in the K-th SCC will create a walk of length L starting at the (K+1)-th SCC. The complexity will not change, the degree of the polynomial bounding the first K SCCs is the same as the degree of the polynomial bounding the first $K+1$ SCCs.

If the (K+1)-th SCC is a cycle, then we have a more interesting case. First of all, there may be multiple nodes that are connected to the K-th SCC. Let's focus our attention to a node in the (K+1)-th SCC, and label it with index 0. We will index the next nodes in the cycle 1, 2, 3,.., etc. Some of these nodes can be exit points that are connected to the K-th SCC:



If node index i is connected with the next SCC, then there are walks of length: $i, i+c, i+2c, i+3c, ...$ that start at node 0 and finish at node i. For each of these walks, we can move from node i to the next SCC. For a total walk length L, Imagine we pick a integer T and spend T moves in SCC $K+1$, then we spend $L-T$ moves in the remaining SCCs. The number of walks in the remaining SCCs will be bounded by some polynomial $P(L-T)$ and there are $O(L)$ ways to pick T. This means that the number of walks with these $K+1$ SCCs will be polynomial, but the degree will increment by 1 from the result for the K SCCs.

This proves that when each SCC is a single node or a cycle, the number of walks is bounded by a polynomial. Incidentally, it also teaches us a method to find the degree of that polynomial, which is the solution to the division 1 version of the problem. In a single walk in the SCC graph, the degree of the polynomial that bounds the number of walks going through those SCCs in the original walks is equal to max([Number of SCCs in the walk that are cycles and not just a single node] - 1, 0). We can then find the walk with the maximum such degree using dynamic programming or memoization , because the SCC graph is a DAG.

Code

In division 2, we only need to verify if each of the graph's SCCs is a single cycle or a node. If not, then the graph is unbounded, else it is bounded.

```
string isBounded(vector<string> graph)
{
    // Find the SCCs of the graph.
    // You can use Tarjan's algorithm to do it in O(n^2) time, but since n <= 50,
    // we can even use Floyd-Warshall to get the reachability graph and then
    // two vertices (i,j) are in the same SCC if you can reach i from j and vice versa
    int n = graph.size();
    int reach[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            reach[i][j] = ( (graph[i][j] == 'Y') ? 1 : 0 );
        }
    }
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                reach[i][j] |= reach[i][k] & reach[k][j];
            }
        }
    }
    vector<bool> saved(n);
    for (int i = 0; i < n; i++) {
        if (! saved[i] ) {
            // Save the SCC:
            vector<int> scc(1, i);
            cout << "SCC: "<<i;
            for (int j = i+1; j < n; j++) {
                if (reach[i][j] == 1 && reach[j][i] == 1) {
                    cout << ", "<<j;
                    scc.push_back(j);
                    saved[j] = true;
                }
            }
```

Alternative solutions and additional comments.

<Place your comments here>

Next problem: MysticAndCandies

By **vexorian**

TopCoder Member

Editorial feedback Part 1	Choose
I liked it.	
I didn't like it.	

Editorial feedback division 1 hard	Choose
I liked it.	
I didn't like it.	

3 Comments | Add Comment