15295 Fall 2019 #07 Network Flow -- Problem Discussion

October 9, 2019

This is where we collectively describe algorithms for these problems. To see the problem statements follow this link. To see the scoreboard, go to this page and select this contest.

A. Network Flow (Easy)

B. Those are not the droids you're looking for

C. Flame of Nucleus

First run floyd-warshall to find the shortest path between each dome. Then we need a node for each dome on the left and right of our network. Connect source to the left nodes with capacity P_i, and right nodes to sink with capacity K_i. Connect a left node to a right node with capacity infinity if and only if the shortest path between them is smaller than L days (two nodes representing the same dome will be connected). Then running maxflow will tell us maximum number of people that can survive. Each pushed flow of capacity k from source to a left node representing ith dome to a right node representing jth dome to the sink means: we allocate k people to travel from ith dome to jth dome.

--Sheng

D. Array and Operations

This question is essentially asking how many fraction reductions we can perform on designated pair of numbers in an array. Our first observation is that dividing by primes is always superior -- we should divide by 2 instead of 4 to get max operations. Our second observation is that within good pairs, one index is odd and the other one is even, so even indices and even indices are never in a good pair, same for odd and odd. Thus we can put evens on the left and odds on the right side of our network. Our third observation is that different primes do not interfere, so can be treated in separate runs (this makes our code cleaner). We first factor all numbers, and collect their prime factors in a set . Then we do the following for each prime:

A single run of our network would look like (say for prime 2): source is connected to all odds, evens are connected to the sink, and an odd index and even index are connected if they are within a good pair. Essentially, pushing 1 flow here means doing a fraction reduction on one of the good pairs. As an example, the capacity of source to odd indexed 24 would be 3 (largest power of 2 that is factor of 24 is 3), and capacity of even indexed 12 to sink would be 2. This means in total, 24 can be reduced 3 times, regardless of which pairs we perform the reduction in. The capacity of the edge from 24 to 12 clearly should be the minimum of 2 and 3.

Then we just sum up the flow of each individual run. Alternatively, the first time I did this I used a dictionary to record the number of primes within each number (so the dictionary for number 24 has 3:1, 2:3). Then I would create 2 nodes for 24 (call them 24_2 and 24_3). If 24 and 12 (which has 12_2 and _12_3) are in a good pair, I would connect (24_2 and 12_2), and (24_3 and 12_3). This also works, and can be done in one single run of dinic, but is harder to do because it could be a pain to keep track of all the node indices in the network. In general, once we see that the 2 nodes, the 3 nodes, and the 5 nodes don't connect to each other, we should realize that it is simpler to separate them into different networks.

--Sheng

E. Fox And Dinner

Intuitively, we want to partition the k numbers into cycles of adjacent primes, such that each cycle has length >= 3. A better way to look at this is we want to eliminate 2-cycles. The central observation is: for 2 distinct numbers to sum to a prime, they must have different parity! In fact, an input can be directly pronounced "impossible" if number of odds and evens don't match, and in our final answer each cycle must have same number of interleaving odds and evens. Thus we can put odd numbers on the left of our network and even numbers on the right. We connect the odds and evens that sum to a prime with capacity 1. We also realize that we want each number to be paired with 2 numbers (its predecessor and successor in the cycle). Thus we connect source to each odd with capacity 2, and each even to sink with capacity 2. This eliminated 2-cycles! We only call the network saturated if the max flow is k (all even edges to sink are saturated).

Finally, to construct the actual cycles from the flow graph, we can run DFS on the numbers. My code was a bit shorter than DFS, because we can play some tricks if we know each number has exactly 2 neighbors.

--Sheng

F. Pool construction
This is a 451 recitation question (go take 451!). Essentially we just use min-cut instead of max-flow. We want the value of a cut to correspond to the cost of some configuration of building the pool. So let each patch be a node in our network, and let the "source" side of the cut represent nodes that will be holes in the end, and let "sink" side represent nodes that will be grass in the end. Then we connect source to each starting hole with capacity f, and connect each starting grass to the sink with capacity d. These represent the cost of conversion. If a starting grass ends up as a grass, it will end up on the "sink" side -- the capacity d edge is not in the cut, corresponding to costing nothing. If a starting grass ends up as a hole, it will end up on the "source" side. Now the capacity d edge must be in the cut, corresponding to the cost of converting grass to hole. We also add edges of capacity b between adjacent patches. If two patches end up both as grass or both as holes in the end, then the edge between them will not be in the cut, corresponding to costing nothing.

One caveat is that the outermost rows and columns must be grass. The way we deal with this is: manually turn all outermost holes into grass (and record the cost) before running maxflow. Also, to ensure that these grasses are never turned into holes, we should make sure the edges between these grasses and the sink are never in the min-cut. Thus we set the capacity of these edges to be infinity.

Running dinics on this network will output the maxflow, which is mincut, which corresponds to min cost.

--Sheng