2019-12-20 Gold Review

Joshua Zhang, Pranav Mathur, and Ray Bai

December 2019

1 Problem 1: Milk Pumping

1.1 Problem Statement

Farmer John has recently purchased a new farm to expand his milk production empire. The new farm is connected to a nearby town by a network of pipes, and FJ wants to figure out the best set of these pipes to purchase for his use in pumping milk from the farm to the town.

The network of pipes is described by N junction points (endpoints of pipes), conveniently numbered $1 \dots N$ ($2 \leq N \leq 1000$). Junction point 1 represents FJ's farm and junction point N is the town. There are M bi-directional pipes ($1 \leq M \leq 1000$), each joining a pair of junction points. The ith pipe costs c_i dollars for FJ to purchase for his use, and can support a flow rate of f_i liters of milk per second.

FJ wants to purchase a single path worth of pipes, where the endpoints of the path are junctions 1 and N. The cost of the path is the sum of the costs of the pipes along the path. The flow rate along the path is the minimum of the flow rates of the pipes along the path (since this serves as a bottleneck for the flow traveling down the path). FJ wants to maximize the flow rate of the path divided by the cost of the path. It is guaranteed that a path from 1 to N exists.

INPUT FORMAT (file pump.in):

The first line of input contains N and M. Each of the following M lines describes a pipe in terms of four integers: a and b (the two different junctions connected by the pipe), c (its cost), and f (its flow rate). Cost and flow rate are both positive integers in the range $1 \dots 1000$.

OUTPUT FORMAT (file pump.out):

Please print 10^6 times the optimal solution value, truncated to an integer (that is, rounded down to the next-lowest integer if this number is not itself an integer).

1.2 Key Observation

The network of pipes can be easily represented by a graph, where the pipes are the edges and the junctions are vertices. If we set the weight of each edge on the graph to the cost it takes FJ to use it, we can run a shortest path algorithm, such as Dijkstras or Bellman-Ford to find the cost of the cheapest path from the farm to the town in $O(M \log N)$ time. If flows did not matter(e.g. all the flows are 1) then the optimal path would just be the cheapest one found by these algorithms. However, when we factor in different flow weights into this problem, we realize the that cheapest path may not always be the best. A path using pipes $(1, 1)$ and $(2, 1)$ (where flow is the first number and cost is the second) is cheaper than a path using $(20, 10)$ and $(20, 9)$, but the second path has a higher flow to cost ratio. So, how do we account for differing edge weights in our problem?

1.3 Solution 1: Deleting Minimum Flow Edges

We notice that, for any given network of pipes, the minimum flow possible for any path in that network is the minimum flow of any individual pipe on the graph. This gives us a lower bound for our flow on our graph. If we remove the edge with the lowest flow, our flow is now bounded by the second lowest flow in our graph.

Going off of this observation and the earlier one in section 2, we can build a solution to solve the problem. Before we begin, we sort our edges in non-decreasing order by flow. We run Dijkstras on our graph to get the cost of the cheapest path, then use this value and the smallest flow value in the graph to compute a ratio of flow to cost. Next, we remove the edge with the lowest flow and repeat the previous step to generate a new ratio. When we reach the point where a path to the town from the farm no longer exists, we stop. We keep a running maximum of flow to cost ratios over this process, and print that as our answer.

Algorithm 1 Deleting Minimum Flow Edges

```
1: sort(edges)
2: max ← 0
3: while path exists do
4:     cost ← dijkstras()
5:     remove lowest flow edge
6:     flow ← lowestflow
7:     if flow/cost > max then
8:         max ← flow/cost
9: print ⌊max · 10^6⌋
```

Since Dijkstras takes $O(M \log N)$ time and we need to run it up to $M-1$ times, we get an overall complexity of $O(M^2 \log N)$, and since $N, M \leq 1000$, this runs in time.

1.4 Solution 2: Modified Dijkstras

To begin, we initialize a matrix $dp[i][j]$ with i and j both being equal to 1000 (the bounds of N and M respectively) and fill it with very large numbers($> 10^6$ will do). Let $dp(i, j)$ represent the minimum cost of reaching vertex i with flow j. When we run Dijkstras, instead of checking whether our cost to reach the vertex is lower than all previous attempts, we check if our cost to reach a vertex with a certain flow value is lower than that of all previous attempts to reach the vertex with the same flow value. Each time we go from any node to the end, we calculate the ratio of that path and update a running maximum of ratios. Note that this means that the "nodes" we put in our PriorityQueue during Dijkstras need to contain the vertex, the cost, and the flow.

Algorithm 2 Modified Dijkstras

```
dp ← int[1000][1000]
for all i < 1000, j < 1000 do
    dp[i][j] ← 10^6 + 1
max ← 0
pq ← priority queue
add starting node to pq
while pq is not empty do
    curr ← poll pq
    if curr's vertex is N then
        max ← max(max, curr's cost/dp[N − 1][curr's flow])
    for all edges connected to curr's vertex do
        con ← connected node
        flow ← min(curr's node, con's node)
        cost ← dp[curr's node][curr's flow] + con's cost
        vert ← con's vertex
        if cost < dp[vert][flow] then
            dp[vert][flow] ← cost
            add(vertex(cost, flow, vert))
print ⌊max · 10^6⌋
```

Dijkstras takes $O(M \log N)$ time and we ran it only once, we get an overall complexity of just $O(FN + M \log N)$(The FN is from making our dp array). This is faster than our previous solution, however, I personally feel like this is less intuitive than the previous solution and slightly trickier to come up with and implement.

2 Problem 2: Milk Visits

2.1 The Problem

Farmer John is planning to build N ($1 \leq N \leq 10^5$) farms that will be connected by $N-1$ roads, forming a tree (i.e., all farms are reachable from each-other, and there are no cycles). Each farm contains a cow with an integer type T_i between 1 and N inclusive.

Farmer John's M friends ($1 \leq M \leq 10^5$) often come to visit him. During a visit with friend i, Farmer John will walk with his friend along the unique path of roads from farm A_i to farm B_i (it may be the case that $A_i = B_i$). Additionally, they can try some milk from any cow along the path they walk. Since most of Farmer John's friends are also farmers, they have very strong preferences regarding milk. Each of his friends will only drink milk from a certain type of cow. Any of Farmer John's friends will only be happy if they can drink their preferred type of milk during their visit.

Please determine whether each friend will be happy after visiting.

In this problem, you are given a tree with cows of different types at each node, and list of queries to perform on the tree. Each query gives you the indices of two nodes and asks you to determine if the shortest path between those two nodes contains at least one cow of a certain type.

2.2 My Solution

We will solve the problem by processing the queries offline (in a different order than they are presented which makes the problem easier to solve). Sort the queries in order of increasing C_i (the type of cow you need to find along the path from A_i to B_i). This allows us to reduce the problem to finding the maximum C_i along the path from A_i to B_i. To do this, we will need to do some preprocessing on the graph. Before we process queries, store every value of C that occurs in the tree with all the locations at which it occurs. Then, as we are going through the sorted queries and encounter new values of C, we can add them to the graph. When we process every query, the values of C in the tree are always less than the C_i that we are currently processing. Because of this, we can query the maximum value in the tree on the path from A_i to B_i and if this value is equal to C_i, then the farmer will be happy.

Of course, now we need a way to efficiently query the maximum value in the tree along the path from A_i to B_i. Fortunately there is a technique called heavy light decomposition that allows us to use a segment tree to perform such queries in $\log N$ time. This solution runs in $O(M \log N)$ time.

2.3 USACO's Solution

In this solution, we will also use offline queries, but we will process them differently. We will solve the queries while doing a DFS on the graph. Before doing the DFS, store a data structure which maps a node to the list of queries for that node. Also, do a preorder traversal of the graph, which will allow us to determine if a node is an ancestor of another node. Now we are ready to answer the queries using a DFS.

While doing the DFS, maintain a stack of nodes on the path from node 1 to the current node x, as well as a list that stores, for each type of cow seen along the path from 1 to x, the nodes along the path that have a cow of that type and the depth of that node. When we process a node, we will answer all queries that have that node as one of their endpoints. We do this by checking if the last farm y corresponding to the type of cow C_i in the query is on the path from A_i to B_i. If y is not an ancestor of B_i, then the path from A_i to B_i contains y and the query is true. Otherwise, for the query to be true, y must be the LCA of A_i and B_i. We can do this by directly calculating the LCA, or by noticing that if the node Y that is one deeper than y is an ancestor of both A_i and B_i, then y is not the LCA. Otherwise, A_i and B_i are in different subtrees of y, so y is the LCA, and the query is true. Once a node is processed, process its children in the same and all the queries will be answered. This solution runs in $O(N + M)$ time.

3 Problem 3: Moortal Cowmbat

3.1 Problem Statement

There are M buttons labeled by the first M lowercase letters ($1 \leq M \leq 26$). Bessie's favorite combo of moves in the game is a length-N string S of button presses ($1 \leq N \leq 10^5$). However, due to the most recent update, every combo must now be made from a series of "streaks", where a streak is defined as a series of the same button pressed at least K times in a row ($1 \leq K \leq N$). Bessie wants to modify her favorite combo to produce a new combo of the same length N, but made from streaks of button presses to satisfy the change in rules.

It takes a_{ij} days for Bessie to train herself to press button j instead of button i at any specific location in her combo (i.e. it costs a_{ij} to change a single specific letter in S from i to j). Note that it might take less time to switch from using button i to an intermediate button k and then from button k to button j rather than from i to j directly (or more generally, there may be a path of changes starting with i and ending with j that gives the best overall cost for switching from button i ultimately to button j).

Help Bessie determine the smallest possible number of days she needs to create a combo that supports the new requirements.

3.2 Solution

- Run Floyd Warshall on cost matrix
- Precompute necessary prefix sums
- Run Dynamic Programming Algorithm

3.3 Dynamic Programming

Let $dp[i][j]$ represent the smallest amount of time needed so that the first i letters create a "valid combo" and the last letter is j (so the last K letters are all j). Our DP states will require that we maintain range minimums in some way. While segment trees are an option, one can realize that a second $dpmin[i]$ array will suffice. $dpmin[i]$ represents the minimum value of all $dp[i][j]$ across all possible j.

dp[i][j] = min(ps[i][j]-ps[i-K][j]+dpmin[i-K],dp[i-1][j]+cst[i][j]);

The $ps[i][j]$ matrix represents prefix sums, whereas the $cst[i][j]$ matrix represents the input cost matrix after Floyd Warshall has been executed on it.

3.4 Time Complexity Analysis

Floyd Warshall will run in $O(M^3)$ because we have M letters. Our DP states require $O(N * M)$ memory and runs in $O(N * M)$ time.

Note that the test cases were structured in such a way that some version of this solution that's missing an optimization will still earn a fair number of test cases.