USACO Silver and Gold Review

Neeyanth Kopparapu and Patrick Zhang

December 5, 2019

Introduction

This lecture will cover a few concepts and approaches we have learned through a series of USACO Gold problems that we picked. Remember that in general, USACO problems are divided into two main categories: those that can be solved through an algorithm you learn in a lecture (MST, DP, DFS, etc.) and ad hoc problems. Because ad-hoc problems don't rely on a concept but rather general programming intellect, we can't really go over tips for these except for:

- Look for patterns!
- Trace through smaller test cases and try to see if there is a common thing you have to do to solve the problem - can it generalize into a solution that can be written with the proper bounds and restrictions?

1 Dynamic Programming

1.1 Snakes

Problem Statement: According to legend, St. Patrick banished all of the snakes in Mooland over a thousand years ago. However, snakes have since made their way back to Mooland! St. Patrick's day was on March 17, so Bessie is going to commemorate St. Patrick by banishing all of the snakes from Mooland once and for all.
Bessie is equipped with a net to capture snakes distributed in N groups on a line ($1 \leq N \leq 400$). Bessie must capture every snake in every group in the order that the groups appear on the line. Each time Bessie captures a group, she can put the snakes in a cage and start with an empty net for the next group.
A net with size s means that Bessie can capture any group that contains g snakes, where $g \leq s$. However, every time Bessie captures a group of snakes of size g with a net of size s, she wastes sg space. Bessie's net can start at any size and she can change the size of her net K times ($1 \leq K < N$).
Please tell Bessie the minimum amount of total wasted space she can accumulate after capturing all the groups.

Solution Sketch: We solve this problem by dynamic programming. Let $dp[m, k]$ be the minimum sum of net sizes needed to catch the first m groups of snakes with k net size changes. Then $dp[m, 0] = m \cdot \max\{a_1, \cdots, a_m\}$ and for all $k > 0$,

$$dp[m, k] = \min_{i<m}(dp[i, k1] + (mi)max\{a_{i+1}, \ldots, a_m\})$$

with the convention that $dp[0, k] = 0$. Then the answer is $dp[N, K] = \sum_{i=1}^{N} a_i$. Naively, there are $O(N^2)$ states each with $O(N)$ transitions each computable in $O(N)$ time, but the resulting complexity of $O(N^4)$ is too slow. However, it can be improved. For each m and k, start with $i = m1$ and decrement down to 0. Then $\max\{a_{i+1}, \ldots, a_m\}$ can be maintained with constant time work for each i, so the cost of computing all transitions for $dp[m, k]$ is only $O(N)$. This yields an $O(N^3)$ runtime, which is sufficient for the given bounds.

1.2 Cow Poetry

Problem Statement: Unbeknownst to Farmer John, Bessie is quite the patron of the arts! Most recently, she has begun studying many of the great poets, and now, she wants to try writing some poetry of her own. Bessie knows N ($1 \leq N \leq 5000$) words, and she wants to arrange them into poems. Bessie has determined the length, in syllables, of each of her words, and she has also assigned them into "rhyme classes". Every word rhymes only with other words in the same rhyme class.
Bessie's poems each include M lines ($1 \leq M \leq 105$), and each line must consist of K ($1 \leq K \leq 5000$) syllables. Moreover, Bessie's poetry must adhere to a specific rhyme scheme. Bessie would like to know how many different poems she can write that satisfy the given constraints.

Solution Sketch:

In order to solve this problem, we must make a series of observations. The first is that it doesn't matter what order the rhyme classes are given - what matters is the frequencies of each rhyme class. Once we compute the frequency of each rhyme class f_i and the number of ways to end a line with a rhyme w_k, the number of poems that bessie can write for that rhyme class is

$$\sum_{k=1}^{num_{rhymes}} w_k^{f_i}$$

The final answer is the answers for all of the rhyme classes multiplied together. For instance, there are 8 ways to end in rhyme 1 and 4 ways to end in rhyme 2. There are 2 of rhyme class A and 1 of rhyme class B. Thus, the answer is

$$(8^2 + 4^2)(8^1 + 4^1) = 960.$$

Now, we need to find the number of ways to form a line that ends in each rhyme. To do this, we use dynamic programming. Let $dp[i]$ be the number of ways to form a line of length i. The transition between states is, for the number of syllables in every word w_s, is:

$$dp[w_s + i] += dp[i].$$

(Don't forget to modulo 1,000,000,007 at every step!)

Whenever $w_s + i == K$, we increment $r[w_r]$, where w_r is the rhyme of that word and r is an array storing the number of ways to form a line that ends in each rhyme.

The efficiency of the dynamic programming is $O(NK)$ and the exponentiation is $O(NM \log M)$ for the worst case, which fits inside the time limit.

2 Graph Theory

2.1 Shortcut

Problem Statement: Every evening, Farmer John rings a giant bell that summons his cows to the barn for dinner. Eager to get to the barn as quickly as possible, they all follow the shortest possible route to get there.
The farm is described by a set of N fields ($1 \leq 10,000$), conveniently numbered $1 \ldots N$, with the barn residing in field 1. The fields are connected by a set of M bidirectional trails ($N1 \leq 50,000$). Each trail has a travel time associated with it, and there is a path from every field to the barn using some set of trails.
Field i contains c_i cows. Upon hearing the dinner bell, these cows all walk to the barn along a route that takes the minimum amount of time. If there are several routes tied for the minimum time, the cows take whichever of these is "lexicographically" smallest (i.e., they break ties between two routes by favoring the one using the lower-indexed field at the first place where the routes differ, so for example a path that visits fields 7, 3, 6, 1 would be preferable to one that visits 7, 5, 1, assuming both had the same travel time).
Farmer John is worried about the barn being far away from some fields. He adds up the travel time experienced by each cow, summed over all the cows, calling this number the total travel time. He would like to reduce this number as much as possible by adding one extra "shortcut" trail which has a travel time

of T ($1 \leq 10,000$), from the barn (field 1) to some other field of his choosing. If a cow stumbles upon the shortcut trail while traveling along her usual path to the barn, she will take it if it gets her to the barn faster. Otherwise, a cow will follow her usual route, even if it might have been possible to use the shortcut to improve her travel time.
Please help Farmer John determine the greatest possible amount of decrease in total travel time he can achieve by adding his shortcut trail.

Solution Sketch: In order to solve this problem, we must compute two things for every vertex: the shortest path to the barn, and the number of cows that pass through that vertex.
Using Dijkstra's algorithm, which computes the minimum path from each field to the barn, will help with both tasks. However, we must also store the parents so that we can backtrack to find the fields on the path to the barn. We also need to check that we are finding the lexicographically smallest shortest paths whenever the current lengths of the paths are equal.
After running Dijkstra's algorithm, we can backtrack from every vertex to record how many cows pass through every field. This will run in $O(N^2)$.
You can also use a dfs on the shortest path tree (the tree generated from Dijkstra's algorithm), which would allow you to accomplish that task in $O(N)$. Simply construct an edge list using the parent array then calculate the sum of the number of cows that pass through fields in its subtree.
Finally, we compute how much distance adding a road from the barn to the vertex saves, which is $\max c_i(d_i)$ for (2), where c_i is the number of cows that passes through field i and d_i is the shortest path from field i to the barn.
Dijkstra's algorithm is $O(M)$ and backtracking is $O(N^2)$, for a total efficiency of $O(M + N^2)$ (or $O(M + N)$ if you use dfs), both of which are fast enough to fit in the time limit.

2.2 Mootube

Problem Statement: In his spare time, Farmer John has created a new video-sharing service, which he names MooTube. On MooTube, Farmer John's cows can record, share, and discover many amusing videos. His cows already have posted N videos ($1 \leq 100,000$), conveniently numbered $1 \ldots N$. However, FJ can't quite figure out how to help his cows find new videos they might like. FJ wants to create a list of "suggested videos" for every MooTube video. This way, cows will be recommended the videos most relevant to the ones they already watch.
FJ devises a metric of "relevance," which determines, as the name suggests, how relevant two videos are to each other. He picks $N1$ pairs of videos and manually computes their pairwise relevance. Then, FJ visualizes his videos as a network, where each video is a node and the $N1$ pairs of videos he manually considered are connected. Conveniently, FJ has picked his $N1$ pairs so that any video can be reached from any other video along a path of connections in exactly one way. FJ decides that the relevance of any pair of videos should be defined as the minimum relevance of any connection along this path.
Farmer John wants to pick a value K so that next to any given MooTube video, all other videos with relevance at least K to that video will be suggested. However, FJ is worried that too many videos will be suggested to his cows, which could distract them from milk production! Therefore, he wants to carefully set an appropriate value of K. Farmer John would like your help answering a number of questions about the suggested videos for certain values of K.
For every query q_i you are given v_i and k_i. Answer how many videos will be suggested to viewers of video v_i if $K = k_i$.
Solution Sketch: We will solve this problem using DSU.
Sort the queries in decreasing order by threshold and start with an empty graph. Loop through the sorted queries. When we reach a certain query q_i, we add all of the edges to the DSU with thresholds greater than or equal to k_i. We can efficiently add all of the edges by sorting the edges in decreasing order.
The answer for that query is the size of the set at v_i. When outputting your answer, don't forget to output the answers in the same order that the queries were given.