15-451/651 Assignment 3

Vy Nguyen
vyn
Recitation: B
February 22, 2018

1: Streaming with Nearly Optimal Space

(a)

Let $p = min(1, 400/(m\epsilon^2))$ be the probability of keeping an element in the stream. Since this is a binomial distribution, the number of occurrences f_i' of item i in the stream has an expected value of $\mathbb{E}(f_i') = f_i * p$ and variance of $Var(f_i') = f_i * p * (1-p)$. So by the Chebyshev's inequality, we get

$$Pr[|f_i' - pf_i| \geq (\epsilon/2)mp] \leq \frac{pf_i(1-p)}{((\epsilon/2)mp)^2} \leq \frac{pf_i}{((\epsilon/2)mp)^2} = \frac{4f_i}{m^2\epsilon^2 p}$$

Now simultaneously for all $i \in [n]$, the probability that $|f_i' - pf_i| \leq (\epsilon/2)mp$ or equivalently, $|f_i'/p - f_i| \leq (\epsilon/2)m$ is

$$Pr[\bigwedge_i |f_i'/p - f_i| \leq (\epsilon/2)m] = \neg Pr[\bigvee_i |f_i'/p - f_i| \geq (\epsilon/2)m] \tag{1}$$

$$\geq 1 - \sum_i \frac{4f_i}{m^2\epsilon^2 p} \tag{2}$$

$$= 1 - \frac{4m}{m^2\epsilon^2 p} = 1 - \frac{4}{m\epsilon^2(\frac{400}{m\epsilon^2})} \tag{3}$$

$$= 1 - \frac{1}{100} = \frac{99}{100} \tag{4}$$

Thus, for all $i \in [n]$, $Pr[|f_i'/p - f_i| \leq (\epsilon/2)m] \geq \frac{99}{100}$

(b)

Apply the majority algorithm to find the majority element, then use the count-min method to hash the elements from the stream using $\lceil \frac{1}{lg(1/11)} \rceil$ hash functions $h : \{1...n\} \to \{1...m\}$ from a family of universal hash functions. Construct a table T of dimension $\lceil \frac{1}{lg(1/11)} \rceil$ by m. For every incoming kept element e, compute $h_i(e)$ and increase the corresponding counters in each row $T[i, h_i(e)]$ for each $i = 1, 2, ..., \lceil \frac{1}{lg(1/11)} \rceil$; the estimated count $\hat{f}_{i*}$ is the minimum count in each column. Return the estimated count for the majority element; if there is no majority element, return anything.

There are two sources of error in this algorithm, one from the sampling and the other from the collisions due to hashing. From part(a), we know that the count before hashing f_i'/p is at most $(\epsilon/2)m$ from the true count f_i with high probability. And from lecture, we know that the count-min method has the probability of at least $1 - \delta$ of making the error $|\hat{f}_{i*} - f_{i*}| \leq \epsilon m$. Since the before-hashing count is at most $(\epsilon/2)m$ from the true count, the after-hashing count also needs to account for this error in addition to its own error. So, since sampling and hashing are independent operations, we can combine the two probabilities and get $Pr(|\hat{f}_{i*} - f_{i*}| + (\epsilon/2)m \leq \epsilon m) \geq \frac{99}{100}(1-\delta)$ which equals to $Pr(|\hat{f}_{i*} - f_{i*}| + \leq (\epsilon/2)m) \geq \frac{99}{100}(1-\delta)$. We can choose $\delta = 1/11$ and get the desired error bound of $Pr(|\hat{f}_{i*} - f_{i*}| + \leq \epsilon m) \geq \frac{9}{10}$.

Since each element is kept with probability p, the space complexity can be reduced by maintaining a dictionary of hashed values of size $m * p = 400/\epsilon^2$ instead of m. In the event that the space exceeds this, we just terminate the algorithm. It takes $O(logn)$ bits to generate the coin toss and store the majority element and its counter, and $\frac{1}{lg(1/11)} log(nm) = O(logn + log(1/\epsilon))$ bits to store all of the hash functions. Maintaining the table of counters requires $O(1/\epsilon)$ bits. So in total, the required space is $O((1/\epsilon)(logn + log(1/\epsilon)) \leq O(logn + (1/\epsilon)(log(1/\epsilon))$.

2: A Colorful Tree

Consider a binary tree T with root r and children A and B. To solve the k-partition problem with minimum split edges, consider every possible way to split k red nodes between the two children. Suppose that A gets k' red nodes and B gets the remaining $k - k'$ red nodes. Now there are four sub-problems to consider:
Case 1: A is red, B is red
Case 2: A is red, B is blue
Case 3: A is blue, B is red
Case 4: A is blue, B is blue
Since the root r can either be red or blue, there are eight cases in total. Let $C(T, col, k)$ be the total weight of the split edges for tree T that has the root r with color col and k red nodes. The recurrence is

$$C(T, r, col, k) = \begin{cases} 0 & \text{if } k = 0 \\ \sum min\{C(A, red, k') + w_{rA} + C(B, red, k - k') + w_{rB}, & \\ C(A, red, k') + w_{rA} + C(B, blue, k - k'), & \text{if } col = blue \\ C(A, blue, k') + C(B, red, k - k') + w_{rB}, & \\ C(A, blue, k') + C(B, blue, k - k')\} & \\ \sum min\{C(A, blue, k') + w_{rA} + C(B, blue, k - k') + w_{rB}, & \\ C(A, blue, k') + w_{rA} + C(B, red, k - k'), & \text{if } col = red \\ C(A, red, k') + C(B, blue, k - k') + w_{rB}, & \\ C(A, red, k') + C(B, red, k - k')\} & \end{cases}$$

where A, B are the left and right children of node r, w_{uv} is the weight of edge (u, v), and $k' \leq k < n$.

The recurrence holds because we are considering all cases for each color of the root and choose the case that minimizes the total weight of the split edges. Also, the algorithm considered all possible ways to split the k red nodes between the two children. We can memoize with a matrix to store the results of the solved subproblems and output the correct partition at the end.
With memoization, each sub-problem takes $O(k)$ times to solve since it requires constant work for each value $k' \in [1...k]$. At every node, there are at worst k problems since we have to consider all values of k'. Thus each node requires $O(k^2)$ work and the algorithm takes $O(nk^2)$ times. Since k is bounded by the total number of nodes n, the running time is at most $O(n^3)$.

3: On the Road

This problem can be solved by using the Bellman-Ford algorithm to find the shortest path that visits i intermediate nodes. This approach works because since G is a DAG and contains no cycle, there is a relationship between the number of edges and nodes on a path. In particular, the number of edges on any path is one more than the number of intermediate nodes. Let $L[v][k]$ represents the length of the shortest path from source s to v using exactly k edges. The recurrence is the same as Bellman-Ford's

$$L[v][k] = \begin{cases} 0 & \text{if } v = s, k = 0 \\ \infty & \text{if } v \neq s, k = 0 \\ min_{edges(j,v)}(L[j][k-1] + len(j,v)) & \text{else} \end{cases}$$

Here, assume that we already know the shortest paths to each of $v's$ immediate predecessors j using $k-1$ edges, then all we have to do is find the minimum of these plus the length from j to v to get the shortest path from s to v. As the algorithm runs, we can also memoize by keeping a matrix P and storing in $P[v][k]$ the node w that produces the minimum path; so that the path can be reconstructed from there. To enforce the constraint of maximum distance D, choose the largest value k such that $P[v][k-1] \leq D$ and use matrix P to reconstruct that path. Since we're just using Bellman-Ford, the algorithm has a running time of $O(nm)$.