Alex Havrilla

Hwk 3

Task 1:

---

1)

Let s be our stream. Set $p = min(1, 400/(m\epsilon^2))$

We show

$$P(\forall i, |f_i'/p - f_i| \leq (\epsilon/2)m) \geq 99/100$$

or equivalently

$$P(\exists i, |f_i'/p - f_i| > (\epsilon/2)m) \leq 1/100$$

Note that we may write $f_i' = \sum_{j=1}^{f_i} 1_j$ where 1_j is 1 if we do not discard the jth occurence of a_i and 0 otherwise. Then $\mathbb{E}f_i' = \sum_{j=1}^{f_i} P(\text{we keep jth occurence}) = \sum_{j=1}^{f_i} p = pf_i$. Further $Var(f_i'/p) = \frac{1}{p^2}Var(f_i') = \frac{1}{p^2}f_i p(1-p) = \frac{f_i(1-p)}{p}$. So then via Chebyshev's inequality:

$$P(|f_i'/p - f_i| > \epsilon m/2) \leq \frac{f_i(1-p)}{p(\epsilon^2 m^2/4)} = \frac{4\epsilon^2 f_i p(1-p)}{400^2} = \frac{4f_i(1-p)}{400m}$$

Now applying union bound gives that

$$P(\exists i, |f_i'/p - f_i| > \epsilon m/2) \leq \sum_{i\in[n]} \frac{4f_i(1-p)}{400m} \leq \frac{1}{100}$$

as desired since the $\sum f_i = m$

2)

We give a 1-pass streaming algorithm finding the majority item with approximation $|\hat{f}_{i^*} - f_{i^*}| \leq \epsilon m$ where m is stream length, succeeding with probability 9/10 and using $O((1/\epsilon)log(1/\epsilon) + log(n))$ space.

Since we know the stream length is some fixed m beforehand, we uniformly randomly choose some subset $\lceil pm \rceil$ of the m elements to keep and drop the rest. Then we use the a universal hash function $h : [n] \to \lceil \frac{c}{\epsilon} \rceil$ for a constant c later to be specified which we will use to keep counts which costs $O(log(n) + log(1/\epsilon))$ bits. Then as we pass through the stream, we run standard misra gries where we keep a single count for the majority element and a majority candidate, incrementing this count every time we see the majority candidate and decrementing when we do not(and replacing with the current item if count reaches 0). This can be done in $O(log(n) + log(m)) = O(log(n))$(since $m \in O(n)$) but may not keep an accurate approximation of the count of the majority item.

To address this approximation we use our hash table whereby when we see an item which was not prediscarded at the start, we increment its count in the hash table. Then our approximation $\hat{f}_{i^*}$ of f_{i^*} will be $h[a_{i^*}]/p$. We must show that this approximation is within ϵm of the true majority count with 9/10 probability. Note immediately that the hash table requires at most $O(\frac{1}{\epsilon} log(\frac{1}{\epsilon})$ space since we have $O(\frac{1}{\epsilon})$ cells and we increment a particular cell at most $O(\frac{1}{\epsilon^2})$(which can be counted in $O(log(1/\epsilon))$) times since we discard all but $mp = \frac{400m}{m\epsilon^2} = \frac{400}{\epsilon^2}$ elements in the stream. Note that this fufills the $O((1/\epsilon)log(1/\epsilon) + log(n))$ space bound.

We now argue $P(|\hat{f_{i^*}} - f_{i^*}| \leq \epsilon m) \geq 9/10$. Note that uniformly randomly discarding all but $\lceil pm \rceil$ implies that a particular stream element a_i is discarded with probability p(approximately, since we take ceiling). From part a we know $P(\forall i : |\hat{f}_i/p - f_i| \leq (\epsilon/2)m) \geq 99/100$ and in particular we know this holds for the majority item, we we have perfect hashing. So

$$P(|\hat{f_{i^*}} - f_{i^*}| \leq \epsilon m) \leq nP(h(i^*) \neq h(j))P$$

Task 2:

---

a)

We solve K-Partition(on max degree 3 trees) with the following:

For first one, state should be DP[node, num red to use, num blue to use prior color]. Reference notes on independent sets for trees for future assistance

Pick a root node r and orient the tree via DFS.

Define $DP[v, a, c]$ to be the minimal K-partitioning of the subtree rooted at v when v has parent colored c with a remaining nodes to color red. Set $s_v = size(T_v)$ where T_v is subtree rooted at v(this can be computed with the DFS). Let $\delta_c(d) = 1$ if $c \neq d$ and 0 otherwise. Finally set $f_{x=0}(t) = \infty$ if $x = 0$ and t otherwise. Our recurrence is $DP[v, a, c] =$

$$min(f_{a=0}(\delta_c(red) + DP[v_1, a-1, red]), f_{s_v - a=0}(\delta_c(blue) + DP[v_1, a, blue]))$$

when v_1 is the only child of v. Othewise $DP[v, a, c] =$

$$\min_{k \in \{0,\ldots,a\}} (min(f_{a=0}(\delta_c(red) + min(DP[v_1, max(k-1, 0), red] + DP[v_2, a-k, red], \\ DP[v_1, k, red] + DP[v_2, max(a-k-1, 0), red])), \\ f_{s_v - a=0}(\delta_c(blue) + DP[v_1, k, blue] + DP[v_2, a-k, blue])))$$

when v_1, v_2 are children of v. Note this already cumbersome formula is not quite correct as is: when $k = 0$ or a we must choose in the case where we color the current node red whether to subtract this from the red allocated to v_1 or v_2. When $k = 0, a$ the max of $k-1, 0$ or $a-k-1$ become negative and we are forced to remove one red from the other recursive call. We have base cases when v has no children $DP[v, a, c] = 1$ if $a = 0$ and $c = red$ or $a = 1$ and $c = blue$ and otherwise 0. Note root r has not parent so it only has state $DP[r, a, red]$ where we do not add 1 if we color it blue. $DP[r, K, red]$ gives the minimal crossing number. To compute the actual partition we can back track in the array.

This formula is more adequately summarized as minimizing over all possible allocations of red colorings to the children of a node v, casing on whether we color v red or blue. We prove its correctness inductively.

We induct on the structure of the tree. Fix arbitrary K. First consider the singleton. Since it has only a root, it returns 0. We now consider the more general case of a node with one child. We choose to color this node red if $K > 0$ and otherwise blue and then take the minimum of both cases, where $DP[v_1, K-1, c]$ is inductively correct. Now consider a node with two children. We take the minimum over all possible splittings of red colorings, and then take the min over coloring the current node red or blue(incrementing by one appropriately). The recursive calls are inductively justified and clearly the minimum K-partition is found by checking all possibilities in this way.

Note that there are at most $O(n^2)$ subproblems with $O(n)$ work being done at each subproblem. Hence the total runtime is $O(n^3)$.

b)

We solve Palindromic Deletions with the following:

Let s be the input string. We maintain a state where $DP[i,j]$ is the minimum number of palindrome deletions required to eliminate the substring from index i of length j. We calculate the reccurence to be $DP[i,j] = 1$, s[i,j] is a palindrome. Otherwise set $a = 1 + DP[i+k, j-2k], 0 \leq k = \sup_{l<j/2}(\{l : s[i] = s[i+j-1], ..., s[i+l] == s[i+j-(l+1)]\})$ or $a = \infty$ if $k < 0$(sup of emptyset is $-\infty$) and $b = \min_{k \in \{1,...,j-1\}}(DP[i,k] + DP[i+k, j-k])$ and take $DP[i,j] = min(a,b)$

with base cases

$DP[i,0] = 0$ and $DP[i,1] = 1$

We argue inductively that this computes the minimal number of palindromic deletions. First note tht the base cases are correct. Assume the reccurence correct for strings of length less than n. Consider arbitrary string of

length n. We case on its form. If it itself is a palindrome then we know it can be deleted with with cost 1. If its extremal characters do not match then we know the extremal characters will not be deleted as part of the same palindrome, and so we recurse on all 2 partitions of the string which inductively are correct and add the minimal results, taking the min of that and ∞. If the extremal characters do match we find the number of paired characters at the ends and compute the DP[i+k,j-2k] +1. Then we take the min over this and possible partitions. This is correct since any possible deletion strategy deletes extremal characters either together or separately. If they are deleted separately, we know inductively we compute correctly by minimizing over parititions. If they are deleted together, we compute the largest palindrome in which they could be deleted together, and add the minimal number of deletions necessary to successfully reach that palindrome. This concludes our induction.

There are $O(n^2)$ subproblems(number of substrings) with $O(n)$ work done at each one(palindrome checking is $O(n)$ and minimization over j terms is at $O(n)$). We can compute k in $O(n)$ time. Hence the algorithm is $O(n^3)$.

$DP[0, n]$ gives the minimial deletion count.

Task 3:

We show that h as defined in the problem statement has the perfect hashing property with probability 9/10.

Proof. Recall that $h : U \to \{0, ..., 10m^2 - 1\}$ via $h(x) = g(x \mod p)$ where g is a universal hash function with range $10m^2$. So then formally we show

$$P(\forall i, j, i \neq j, h(a_i) \neq h(a_j)) = 1 - P(\exists i, j, h(a_i) = h(a_j))$$

so we want to show $P(\forall i, j, i \neq j, h(a_i) \neq h(a_j)) \leq 1/10$. Via union bound and uniformity we know

$$P(\exists i, j, h(a_i) = h(a_j)) \leq \binom{m}{2} P(h(a_1) = P(h(a_2)))$$

Further

$$P(h(a_1) = P(h(a_2))) = P(g(a_1 \mod p) = g(a_2 \mod p)) =$$

$$P(a_1 \mod p = a_2 \mod p) P(g(a_1 \mod p) = g(a_2 \mod p) | a_1 \mod p = a_2 \mod p) +$$
$$P(a_1 \mod p \neq a_2 \mod p) P(g(a_1 \mod p) = g(a_2 \mod p) | a_1 \mod p \neq a_2 \mod p)$$

Given that $a_1 \mod p$ and $a_2 \mod p$ distinct, we know since g universal that

$$P(g(a_1 \mod p) = g(a_2 \mod p) | a_1 \mod p \neq a_2 \mod p) \leq \frac{1}{10m^2}$$

We also know that

$$P(a_1 \mod p = a_2 \mod p) \leq \frac{log(n)}{10m^2 log(n)}$$

since the number fprimes that divide the difference $|a_1 - a_2|$ is no more than $log(n)$ and we chose p from among the $10m^2 log(n)$ primes(so this is a generous upper bound).

Putting these estimates together yields

$$P(h(a_1) = P(h(a_2))) \leq \frac{log(n)}{10m^2 log(n)} + \frac{1}{10m^2} = \frac{2}{10m^2}$$

so then

$$P(\exists i, j, h(a_i) = h(a_j)) \leq \binom{n}{2} P(h(a_1) = P(h(a_2))) \leq \binom{m}{2} \frac{2}{10m^2} \leq \frac{1}{10}$$

showing the desired bound.

□