CS 170 Homework 12

Due **4/25/2022, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write "none".

2 Streaming Integers

In this problem, we assume we are given an infinite stream of integers x_1,x_2, ..., and have to perform some computation after each new integer is given. Since we may see many integers, we want to limit the amount of memory we have to use in total. For all of the parts below, give a brief description of your algorithm and a brief justification of its correctness.

(a) Show that using only a single bit of memory, we can compute whether the sum of all integers seen so far is even or odd.

(b) Show that we can compute whether the sum of all integers seen so far is divisible by some fixed integer N using $O(\log N)$ bits of memory.

(c) Assume N is prime. Give an algorithm to check if N divides the product of all integers seen so far, using as few bits of memory as possible.

(d) Now let N be an arbitrary integer, and suppose we are given its prime factorization: $N = p_1^{k_1} p_2^{k_2} \dots p_r^{k_r}$. Give an algorithm to check whether N divides the product of all integers seen so far, using as few bits of memory as possible. Write down the number of bits your algorithm uses in terms of $k_1, \dots, k_r$.

Solution:

(a) We set our single bit to 1 if and only if the sum of all integers seen so far is odd. This is sufficient since we don't need to store any other information about the integers we've seen so far.

(b) Set $y_0 = 0$. After each new integer x_i, we set $y_i = y_{i-1} + x_i \mod N$. The sum of all seen integers at step i is divisible by N if and only if $y_i \equiv 0 \mod N$. Since each y_i is between 0 and $N-1$, it only takes $\log N$ bits to represent y_i.

(c) We can do this with a single bit b. Initially set $b = 0$. Since N is prime, N can only divide the product of all x_is if there is a specific i such that N divides x_i. After each new x_i, check if N divides x_i. If it does, set $b = 1$. b will equal 1 if and only if N divides the product of all seen integers.

(d) We can do this with $\lceil \log_2(k_1+1) \rceil + \lceil \log_2(k_2+1) \rceil + \dots + \lceil \log_2(k_r+1) \rceil$ bits. For each i between 1 and r, we track the largest value $t_i \le k_i$ such that p^{t_i} divides the product of all seen numbers. We start with $t_i = 0$ for all i. When a new number m is seen, we find

the largest t_i' such that $p^{t_i'}$ divides m and set $t_i = \min\{t_i' + t_i, k_i\}$ We stop once $t_i = k_i$ for all i as this implies that N divides the product of all seen numbers.

3 Era of Ravens

(a) Design an algorithm that takes in a stream $z_1, \ldots, z_M$ of M integers in $[n]$ and at any time t can output a uniformly random element in $z_1, \ldots, z_t$. Your algorithm may use at most polynomial in $\log n$ and $\log M$ space. Prove the correctness and analyze the space complexity of your algorithm. Your algorithm may only take a single pass of the stream. *Hint:* $\frac{1}{t} = 1 \cdot \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4} \cdots \frac{t-1}{t}$.

(b) For a stream $S = z_1, \ldots, z_{2n}$ of $2n$ integers in $[n]$, we call $j \in [n]$ a *duplicate element* if it occurs more than once.

Prove that S must contain a duplicative element, and design an algorithm that takes in S as input and with probability at least $1 - \frac{1}{n}$ outputs a duplicative element. Your algorithm may use at most polynomial in $\log n$ space. Prove the correctness and analyze the space complexity of your algorithm. Your algorithm may only take a single pass of the stream.

Hint: Use $\log n$ *copies of the algorithm from part a to keep track of a random subset of the elements seen so far. For proof of correctness, note that there are at most* n *indices* t *such that* $z_t \neq z_{t'}$ *for any* $t' > t$*, i.e. element* z_t *never occurs after index* t*.*

Solution:

(a) **Main idea:** Maintain a counter n_1, initially 0, to keep track of how many elements have arrived so far and maintain a "current element" x, also initially 0. When an element z arrives, increment n_1 by 1 and replace x with z with probability $1/n_1$. When queried at any time, output x.

Proof of correctness: To analyze the probability that z_t is outputted at time $t' > t$, observe that z_t must be chosen and then never replaced. This happens with probability

$$\left(\frac{1}{t}\right) \cdot \left(\frac{t}{t+1} \cdot \frac{t+1}{t+2} \cdots \frac{t'-1}{t'}\right).$$

Space complexity: The counter and current element never exceed M and n respective, so we just need $O(\log M + \log n)$ bits of memory.

(b) **Main idea:** The algorithm is to maintain $\log n$ independent instantiations of the sampling algorithm from part (a). When a new element z arrives at time t, first query all the instantiations and if any of them outputs z, ignore the rest of the stream and output z as the 'duplicative element' at the end of the stream. Otherwise, stream z as input to each of the independent instantiations of the sampling algorithm and continue. If the stream ends without the algorithm ever outputting a value, output 'failed' at the end.

Proof of correctness: Note that S must contain a duplicative element by the pigeonhole principle, and if the algorithm returns an element, it is certainly a duplicative element.

We now turn our attention to upper bounding the probability that the algorithm returns 'failed'. Suppose our algorithm returned 'failed', and for each instance of part a, let t be the index of the element it sampled. Note that if z_t is a duplicate of element $z_{t'}$ where $t' > t$, then we would have output z_t when we processed $z_{t'}$. So in order for us to output 'failed' z_t not be a duplicate of any element appearing afterwards in the stream. By part a, t is distributed uniformly at random, so this happens with probability at most $n/2n = 1/2$ per the hint. In turn, the $\log n$ instances collectively give us at most $(1/2)^{\log n} = 1/n$ chance of outputting 'failed'.

Space complexity: We use $\log n$ copies of the data structure from part a, each using $O(\log n)$ bits, so we use $O(\log^2 n)$ bits of memory.

4 Defense Towers

Please answer the following questions after reading the project specifications. **Only for the sake of this entire question assume the following input parameters:** $D = 5$, $5 \leq N \leq 10$, $R_s = 1$, $R_p = 2$

1. Consider the input provided below.

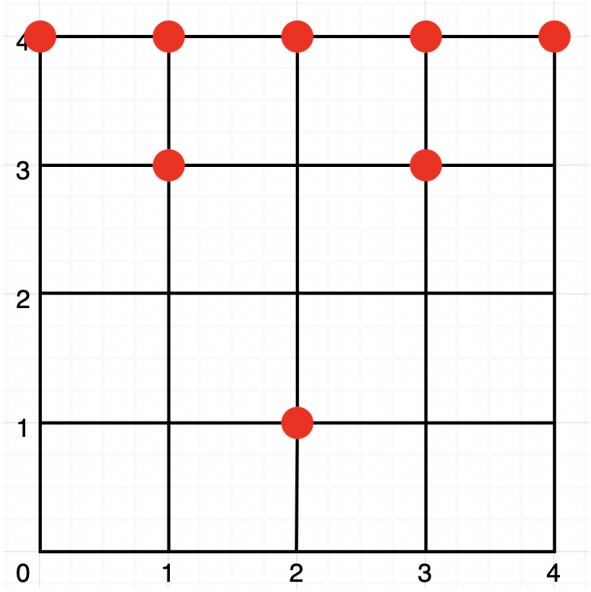

Figure 1: Visualization of Input

The red dots represent cities, i.e. cities are present at the following coordinates: $(0, 4)$, $(1, 3)$, $(1, 4)$, $(2, 1)$, $(2, 4)$, $(3, 3)$, $(3, 4)$, and $(4, 4)$. Using the output format provided in the spec, write the optimal output for this input. Also compute and state the penalty value based on your optimal output.

Solution:

3

1 4

3 4

2 0

Penalty = 170 + 201.502 + 201.502 = 573.004

2. Consider a simple greedy algorithm that picks a tower location that covers the most uncovered cities so far and places a tower at that location. Construct a sample input that would cause this greedy algorithm to give a sub-optimal output. Your input can be simply in the form of a diagram representing cities on a 5 x 5 grid, as long as the coordinates with cities are clearly distinguished. In your answer include the solution that the greedy algorithm would come up with, the penalty value from that solution, the optimal solution, and the penalty value from that solution.

Solution: This is one possible example that would make the greedy algorithm suboptimal.