CSC263 A4

Nena Harsch and Natalia Tabja

October 2024

Question 1 (a) (written by Natalia, verified by Nena)

1. **What is the probability that exactly k iterations of the `while` loop are executed?**

   In each iteration of the loop, the algorithm performs two operations: checking whether $A[i] = x$ and flipping the coin. If $A[i] \neq x$ (which will always be true since no element of A equals x), the algorithm flips the coin to decide whether to stop (return "Not Found") or continue.

   The probability of returning "Not Found" after exactly k iterations is the probability that the algorithm runs $k-1$ times without getting a "Head" on the coin, and then finally gets a "Head" on the k-th iteration. The probability that the coin flip gives TAIL (i.e., continue) is $1-p$.

   Thus, the probability of exactly k iterations is:

   $$P(\text{exactly } k \text{ iterations}) = (1-p)^{k-1} \cdot p$$

   This is because the first $k-1$ iterations must all result in TAIL (probability $1-p$), and the k-th iteration must result in HEAD (probability p).

2. **What is the expected number of iterations made by the `while` loop?**

   The expected number of iterations is determined by the fact that each iteration behaves like a geometric distribution with success probability p. In a geometric distribution, the expected number of trials until the first success is given by:

   $$E[\text{iterations}] = \frac{1}{p}$$

   Thus, the expected number of iterations made by the `while` loop is $\frac{1}{p}$.

Question 1 (b) (written by Natalia, verified by Nena)

What is the probability that exactly k iterations of the `while` loop are executed?

In this case, the algorithm could either: 1) Find the element x on the k-th iteration (with probability $\frac{1}{n}$), or 2) Fail to find x on the first $k-1$ iterations and flip HEAD on the k-th iteration to stop the search.

Let X be the event where the algorithm finds x at index i, and let Y be the event that the coin gives HEAD. The probability of stopping after exactly k iterations is the sum of the probabilities of these two mutually exclusive cases:

$$P(\text{exactly } k \text{ iterations}) = \left(\frac{n-1}{n} \cdot (1-p)\right)^{k-1} \cdot \left(\frac{1}{n} + p \cdot \frac{n-1}{n}\right)$$

The first $k-1$ iterations must result in picking the wrong index ($A[i] \neq x$) and getting TAIL on the coin flip. Then, on the k-th iteration, the algorithm either finds x with probability $\frac{1}{n}$, or gets HEAD on the coin flip with probability p (if $A[i] \neq x$).

Question 2 (written by both, verified by both)

(a) Finding Pr[x = i]

To determine $P[x = i]$, note that the algorithm returns $x = i$ if coin c_i gives HEAD, and all the flips following on coins $c_{i+1}, c_{i+2}, \ldots, c_n$ give TAIL. We do not care about the flips before the i-th coin because whether they change x or not does not matter. We just care that the i-th iteration changes x to i and that the next flips do not change x again. The probability of this happening is:

$$P[x = i] = p_i \prod_{j=i+1}^{n} (1 - p_j)$$

This represents the probability that the i-th flip results in HEAD and all the next flips result in TAIL.

(b) Finding p so that Pr[x = i] = 1/n

We want the value x returned by the randomized algorithm to be such that $P[x = i] = \frac{1}{n}$ for every $1 \leq i \leq n$, i.e., we want x to be a uniform sample from $\{1, \ldots, n\}$.

Consider the probabilities for $n = 3$:

- For $P[x = 1]$, the algorithm returns $x = 1$ if the first coin flip gives HEAD, the second coin flip gives TAIL, and the third coin flip gives TAIL. Therefore:

$$P[x = 1] = p_1 \cdot (1 - p_2) \cdot (1 - p_3) = \frac{1}{3}$$

- For $P[x = 2]$, the algorithm returns $x = 2$ if the second coin flip gives HEAD and the third coin gives TAIL (we do not care about what the first coin gives). Therefore:

$$P[x = 2] = p_2 \cdot (1 - p_3) = \frac{1}{3}$$

- For $P[x = 3]$, the algorithm returns $x = 3$ if the third coin flip gives HEAD (probability p_3) (we do not care about the first two coins). Therefore:

$$P[x = 3] = p_3 = \frac{1}{3}$$

- Solving for p_2:

$$p_2 \cdot (1 - p_3) = \frac{1}{3}$$

Substitute $p_3 = \frac{1}{3}$:

$$p_2 \cdot \left(1 - \frac{1}{3}\right) = \frac{1}{3}$$

$$\frac{2}{3} \cdot p_2 = \frac{1}{3}$$

$$p_2 = \frac{1}{2}$$

- Solving for p_1:

$$p_1 \cdot (1 - p_2) \cdot (1 - p_3) = \frac{1}{3}$$

Substitute $p_3 = \frac{1}{3}$ and $p_2 = \frac{1}{2}$:

$$\left(1 - \frac{1}{3}\right) \cdot \left(1 - \frac{1}{2}\right) \cdot p_1 = \frac{1}{3}$$

$$\frac{2}{3} \cdot \frac{1}{2} \cdot p_1 = \frac{1}{3}$$

$$\frac{1}{3} \cdot p_1 = \frac{1}{3}$$

$$p_1 = 1$$

General Derivation:

By following this pattern, we generalize the probabilities $p_1, p_2, \ldots, p_n$ for any n.

For each i, we can express p_i based on the need to have the coins following the i-th flip give TAIL and the i-th flip give HEAD.

- For $P[x = i]$, the algorithm returns $x = i$ if the i-th coin flip gives HEAD (probability p_i) and the following coin flips all give TAIL (with probabilities $1 - p_{i+1}, 1 - p_{i+2}, \ldots, 1 - p_n$) and. Therefore:

$$P[x = i] = (1 - p_n) \cdot \ldots (1 - p_{i+2}) \cdot (1 - p_{i+1}) \cdot p_i = \frac{1}{n}$$

Now, based on the pattern seen from the $n = 3$ example, we can generalize that for each i, $p_i = \frac{1}{i}$.

Thus, the probabilities that ensure uniform sampling are:

$$p_i = \frac{1}{i}, \quad \text{for each } 1 \leq i \leq n$$

This ensures that $P[x = i] = \frac{1}{n}$ for all i, as required.

Question 3 (a) (written by Nena, verified by Natalia)

Based on the problem, we will utilize disjoint sets to determine if the list L of m items is incorrect. The concept is as follows:

1. Make all the bones singletons, where the distinct integer is the set representative for each bone. We will also have a counter $s = n$, initialized to n because each bone is currently it's own species. The counter represents how many unique species there are.

2. We will traverse through the list L and only check for $S(i, j)$ items, which represent two bones i and j that are from the same species. We will perform a UNION using Weighted Union and Path Compression. Each time we perform a UNION, we also have to decrease the counter s because we are saying that two species/ sets that were previously distinct are now the represented by the same set representative, therefore shrinking the number of unique species by 1.

3. We will traverse through the list L again, this time checking the $D(k, l)$ items, which represent two bones k and l that are from different species. We will perform a FIND on k and l, and if they return different set representatives, then we continue to check the list. However, if they both return the same representative, then we know that the list L is incorrect because two bones that should be different species are represented by the same set representative, meaning they are actually from the same species. We will thus return ERROR FOUND in this case.

4. If we finish checking all the D's without any error, then we will return our counter s, which is the maximum possible number of different species the bones are from.

Question 3 (b) (written by Nena, verified by Natalia)

Algorithm 1 CheckList(B, L):

Input: B: a list of the bones labeled by a distinct integer in {1, 2 ... n} , L: list of m items of the form $S(i, j)$ or $D(k, l)$
Output: Return ERROR FOUND if the list is incorrect. Else, return an integer k, where k is the maximum number of distinct integers.

$species = n$

for each $b \in B$ **do**
    MakeSet(b)
end for

for each $S(i, j) \in L$ **do**
    $s_i \leftarrow$ FindSet(i)
    $s_j \leftarrow$ FindSet(j)
    if $s_i \neq s_j$ **then**
        Union(s_i, s_j)
        $species = species - 1$
    end if
end for

for each $D(k, l) \in L$ **do**
    $s_k \leftarrow$ FindSet(k)
    $s_l \leftarrow$ FindSet(l)
    if $s_k == s_l$ **then**
        return ERROR FOUND
    end if
end for
return *species*

Question 3 (c) (written by Nena, verified by Natalia)

We know that using the forest structure for Union-Find with weighted union and path compression allows us to perform Unions in $O(1)$ time and Finds in $O(log * n)$.

1. We start with a list of bones B that has n distinct bones, and we make them singletons. This takes $O(1)$ time for each bone, so this step takes $O(n)$ time.

2. From above, we can conclude that merging bones of the same species will take $O(log * n + log * n + 1)$, since the merging uses two Find operations and one Union operation. This reduces to $O(log * n)$. Let L have $x \leq m$ items that are of the form $S(i, j)$, so grouping together the elements of this form takes $O(xlog * n)$.

3. From above, we can conclude that checking that two bones are from different species will take $O(log * n + log * n)$, which reduces to $O(log * n)$. If we have x items that are of the form $S(i, j)$, then the remaining $m - x$ items are of the $D(i, j)$. Thus, this step takes $O((m - x)log * n)$.

4. Connecting 2 and 3, these steps will take $O(mlog * n)$ time. Note, that $log * n$ grows at a very slow rate, therefore can be considered as a constant. Thus, we can even say that these steps take $O(m)$ time.

5. Putting it all together, this algorithm will take $O(n + m)$ time, and since $m \geq n$, it simplifies to $O(m)$.