Homework Assignment #4

Quan Xu, Zijin Zhang

Due February 28, 2019, by 5:30 p.m.

Question 1.

Zijin Zhang wrote the solution to this question and Quan Xu read this solution to verify its clarity and correctness.

(a)

The algorithm will return $TRUE$ in the first iteration if and only if the first i it picks satisfies $A[i] = x$. As it given, we know that there are k copies of x in $A[1, ..., n]$. So the probability of finding an x is $\frac{k}{n}$.

(b)

Since the algorthm will either return $TRUE$ or return $FALSE$, the probability of returning $TRUE, P_{TRUE}$ is equal to $1 - P_{FALSE}$ that P_{FALSE} represent the probability of returning $FALSE$. Now consider the scenario that the algorithm will return $FALSE$. The algorithm will return $FALSE$ if and only if it fail to find x in the first r number of trials. We know that the probability of finding x in each trail is $\frac{k}{n}$, so the probability of not finding x in each trail is $1 - \frac{k}{n}$ and we can get the probability of not finding x for r trails is $(1 - \frac{k}{n})^r$. Therefore, the probability of returning $TRUE$ is $1 - (1 - \frac{k}{n})^r$.

(c)

For this part, we want to find the number of iteration for finding x in $A[1, ..., n]$. To be more specific, we want to find the number of failures before the first success, which can be consider as a geometric distribution. Let r be the number of fail iteration until we finally find x in $A[1, ..., n]$ for the first time. As we proved in part (a), the probability of finding x in each iteration is $\frac{k}{n}$. So we have $r \sim Geo(\frac{k}{n})$.
Now, we want to find the expected value of r. Since we know that $r \sim Geo(\frac{k}{n})$,

$$E(r) = \frac{1 - \frac{k}{n}}{\frac{k}{n}} = \frac{\frac{n-k}{n}}{\frac{k}{n}} = \frac{n-k}{k}$$

So the expected number of failures is $\frac{n-k}{k}$. Therefore, the total number of iteration with the last success iteration is $\frac{n-k}{k} + 1 = \frac{n}{k}$.

Question 2.

Quan Xu wrote the solution to this question and Zijin Zhang read this solution to verify its clarity and correctness.

Let $i, j \in \mathbb{N}, 1 <= i \neq j <= n$
Assume $n, m \in \mathbb{N},\ n >= 1$ and $m >= n$.

We will use disjoint set(forest structure) and the operations $Union()$ with WU(weight union by size) and $Find\_set()$ with PC(path compression) to design the algorithm:

- Step 1: Loop over all n distinct variables, $x_1, x_2, ..., x_n$. For i^{th} loop iteration, make a copy of variable x_i and using $Make\_set()$ operation to make each variable as a singleton set, and sets each variable as the representative of its singleton set for now, and sets the size of each set as 1 for now.

- Step 2: Loop over all the m constraints and only do the following instructions for all **equality constraints**:

- For each *equality* constraints, find the representative of the two variables, x_i and x_j, of each *equality* constraints by using $Find\_set()$ operation with PC to find the sets that contains x_i and x_j respectively, and their representative of their own sets respectively. Then union the two sets by using $Union()$ with WU, iff the two representatives are different. After we union the two sets, the new set we created by $Union()$ with Wu has size = sum of the two size of two old sets, and the new representative is one of the two old representatives who has larger size.

- Step 3: Loop over all the m constraints and only do the following instructions for all **inequality constraints**:

- For each *inequality* constraints, we have two variables x_i, x_j. Using the $Find\_set()$ operation with PC to find the two sets that contains x_i and x_j respectively, and their representatives of their own set respectively. If the two representatives are the same, then the algorithm return NIL. Otherwise, continuous.

- Step 4: If no NIL return, then print an assignment of integers to variables$(x_1, x_2, ..., x_n)$.

<u>Now consider the running time of the algorithm in the worst case:</u>

Assume $a, b, c, d \in \mathbb{R}^+$.

- Step 1: There are n distinct variable and $n >= 1$, $Make\_set()$ of each variable takes constant time. And assigning representative and updating size take also constant time. There are n variables, so total runtime is $nC \in \mathcal{O}(n)$.

- Step 2: For each loop iteration, finding each set that contains the variables and the representative of the set takes $\mathcal{O}(log^*n)$ by using $Find\_set()$ operation with PC when it is an *equality* constraints. And the $Union()$ operation with WU takes $\mathcal{O}(1)$, and at most $n-1$ $Union()$ operations. There are at most m iterations of loop, therefore, in total $m(2alog^*n + C) \in \mathcal{O}(mlog^*n)$.

- Step 3: For each iteration of loop, finding the two set that contains $x_i,\ x_j$ by using $Find\_set()$ with PC takes $\mathcal{O}(log^*n)$ OR it will return NIL which takes $\mathcal{O}(1)$. So for worst-case, we never run "return NIL". And the loop will at most iterates m times. So in total runtime is $m(2alog^*n + C) \in \mathcal{O}(mlog^*n)$.

- Step 4: To get each value of the each variable and print it take constant time, and there are n variables. So in total $nC \in \mathcal{O}(n)$.

Therefore, total runtime of worst-case scenario is :
$(an + bmlog^*n + cmlog^*n + dn + C) \in \mathcal{O}(n + mlog^*n) \in \mathcal{O}(mlog^*n)$ (# since $m >= n$) $\in \mathcal{O}(mn)$
So the algorithm is asymptotically better than $\mathcal{O}(mn)$.