CSC263 Assignment 4

Angela Zhu, Xinyi Ji, Zhuozi Zou

March 4, 2019

1 Problem 1

a. Assuming $0 \leq k \leq n$:
The probability that the algorithm returns TRUE in the first iteration is $\frac{k}{n}$, since there are k copies of x in the n integers, and the probability of picking each copy of x in the first iteration is $\frac{1}{n}$.

b. Assuming $0 \leq k \leq n$:
Since the algorithm only returns 2 results - either TRUE or FALSE, the probability that the algorithm returns TRUE is therefore 1 minus the probability that the algorithm returns FALSE.

$$\begin{aligned} & 1 - P(\text{algorithm returns FALSE}) \\ =& 1 - (P(x \neq A[i] \text{ in one iteration}))^{\#\text{ of iterations}} \\ =& 1 - (1 - \frac{k}{n})^r \\ =& 1 - (\frac{n-k}{n})^r \end{aligned}$$

Therefore, the probability that the algorithm returns TRUE is $1 - (\frac{n-k}{n})^r$.

c. The expected number of loop iterations can be calculated by multiplying each possible number of loop iterations by the likelihood that it will occur, and summing all these results up.
When $k = 0$, the loop will go on forever, which means that the "expected" number of loop iterations would be infinite.
When $k = n$, the loop will definitely return TRUE on the first iteration, and the expected number of loop iterations is 1. When $0 < k < n$:
Let i be the number of iterations.
Take $p = P(\text{return TRUE in one iteration}) = \frac{k}{n}$.

$$
\begin{aligned}
& p \cdot 1 + (1-p) \cdot p \cdot 2 + (1-p)^2 \cdot p \cdot 3 + ... \\
=& \sum_{i=1}^{\infty} (1-p)^{i-1} \cdot p \cdot i \\
=& p \sum_{i=1}^{\infty} (1-p)^{i-1} \cdot i \\
=& p \sum_{i=0}^{\infty} (1-p)^{i-1} \cdot i \qquad \text{[without loss of generality]} \\
=& p \cdot [\frac{d}{dp}(-\sum_{i=0}^{\infty}(1-p)^k] \\
=& p \cdot \frac{d}{dp}(-\frac{1}{p}) = p \cdot \frac{1}{p^2} \\
=& \frac{1}{p} = \frac{n}{k}
\end{aligned}
$$

Therefore, the expected number of loop iterations of this algorithm is $\frac{n}{k}$.

2 Problem 2

Algorithm Description

The data structure I am going to use here is the disjoint forest, under the weighted union (by size) rule. The algorithm is as of the following:
First, go through the n distinct variables, let each of them be a set with the set representative being each variable itself.
Second, go through the m constraints, if it is an equality constraint, perform the following steps:

- suppose the form of the constraint is $x_i = x_j$ where $1 \leq i \neq j \leq n$ (as described in the question)
- Find(x_i) and let the result be r_{x_i}
- Find(x_j) and let the result be r_{x_j}
- If $r_{x_i} \neq r_{x_j}$, then we use weighted union to get the union of the two sets, union(r_{x_i}, r_{x_j}), which is represented by r_{x_i} and r_{x_j}. Otherwise we do nothing.

Third, go through the m constraints, if it is an inequality constraint, perform the following steps:

- suppose the form of the constraint is $x_i \neq x_j$ where $1 \leq i \neq j \leq n$ (as described in the question)
- Find(x_i) and let the result be r_{x_i}
- Find(x_j) and let the result be r_{x_j}
- If $r_{x_i} = r_{x_j}$, then we end the algorithm and return NIL, otherwise we do nothing.

Fourth, if we reach this step, then the algorithm has not ended yet, and there is a possible assignment of integers to the variables that can satisfy all the constraints in the list, and we perform the following steps:
Let $i = 0$. Go through the n variables and do the following to each variable:

- suppose the variable is x

- Find(x) and let the result be r_x
- If r_x has been given a value, then we assign x to be equal to r_x, otherwise we assign both r_x and x to be i and we increase i by 1.

Fifth, the assignment has been completed, and we iterate through all the variables (from 1 to n). For each of them, we print out the value we assigned to the variable in the fourth step.

Time Complexity

First step:
Since there are n iterations, and in each iteration we let each of variable be a set which is of $\mathcal{O}(1)$, hence the total time is $n \cdot \mathcal{O}(1)$ which is $\mathcal{O}(n)$.

Second step:
Since we go through the m constraints, there are m iterations.
For inequality constraints, we do nothing.
For each of the equality constraints:

- Find(x_i) is $\mathcal{O}(1$ + length of the Find path) (from lec11 slide, since the data structure is a disjoint forest), which is $\mathcal{O}(1 + \log n)$ (since the disjoint forest is under weighted union by size, from lec11 slide we know the max length of the Find path is $\log n$). Thus Find(x_i) is $\mathcal{O}(\log n)$.
- Same as Find(x_i), Find(x_j) is $\mathcal{O}(\log n)$.
- Comparing r_{x_i} and r_{x_j} is $\mathcal{O}(1)$.
- Doing the weighted union is $\mathcal{O}(1)$ (from lec11 slide, since the data structure is a disjoint forest under weighted union by size).

In the worst-case, the weighted union is needed for each equality constraint and there are at most m equality constraints, the second step is $\mathcal{O}(m \cdot (\log n + \log n + 1 + 1))$, which is $\mathcal{O}(m \cdot \log n)$.

Third step:
Since we go through the m constraints, there are m iterations.
For equality constraints, we do nothing.
For each of the inequality constraints:

- Same as the second step, Find(x_i) is $\mathcal{O}(\log n)$.
- Same as the second step, Find(x_j) is $\mathcal{O}(\log n)$.
- Comparing of r_{x_i} and r_{x_j} is $\mathcal{O}(1)$.
- Returning NIL is $\mathcal{O}(1)$.

In the worst-case, the algorithm does not end and there are at most m inequality constraints, the third step is $\mathcal{O}(m \cdot (\log n + \log n + 1 + 1))$, which is $\mathcal{O}(m \cdot \log n)$.

Fourth step:
Let $i = 0$ is $\mathcal{O}(1)$.
Since we go through the n variables, there are n iterations.
For each iteration:

- Same as the second step, Find(x) is $\mathcal{O}(\log n)$.
- Assigning r_x (if needed) and x is $\mathcal{O}(1)$.
- Increasing i by 1 is $\mathcal{O}(1)$.

Since there are n iterations, the fourth step is $\mathcal{O}(n \cdot (\log n + \log n + 1))$, which is $\mathcal{O}(n \cdot \log n)$.

Fifth step:
Since we go through the n variables, there are n iterations.
For each iteration, printing each variable's value is $\mathcal{O}(1)$.
Since there are n iterations, the fifth step is $\mathcal{O}(n)$.

Therefore, the total time complexity of this algorithm is $\mathcal{O}(n + m \log n + m \log n + n \log n + n)$, which is $\mathcal{O}(2n + 2m \log n + n \log n)$.
For $m, n \geq 0$ and $m \geq n$:

$$2n + 2m \log n + n \log n \leq 2m + 2m \log n + m \log n \leq 2m \log n + 3m \log n$$

Then the total time complexity of this algorithm is $\mathcal{O}(m \log n)$.
For $m, n \geq 0$ and $m \geq n$, $\mathcal{O}(\log n)$ is asymptotically better than $\mathcal{O}(n)$. This means $\mathcal{O}(m \log n)$ is asymptotically better than $\mathcal{O}(mn)$.

Therefore, the worst-case running time of our algorithm is asymptotically better than $\mathcal{O}(mn)$.