CSC263H1 Assignment 4

Jiatao Xiang, Xu Wang, Huakun Shen

February 28th, 2019

Question 1

Written by Huakun Shen; Checked by Jiatao Xiang and Xu Wang

a. There is a probability of $\frac{k}{n}$ for the algorithm to return **TRUE** in the first iteration.
There are n integers in total, where k of them are x. Picking one from them yields $\frac{k}{n}$ of probability to get an x.

b. Geometric Distribution. The probability for the algorithm to return **TRUE** in r iterations is,

$$\begin{aligned} P(x \leq r) &= (1-\frac{k}{n})^0 \cdot \frac{k}{n} + (1-\frac{k}{n})^1 \cdot \frac{k}{n} + ... + (1-\frac{k}{n})^{r-1} \cdot \frac{k}{n} \\ &= \Sigma_{i=0}^{r-1}((1-\frac{k}{n})^i \cdot \frac{k}{n}) \end{aligned}$$

Where x is the number iterations where the first success occurs.

c. When the algorithm is modified with an infinite loop, the expected number of loop iterations is $\frac{n}{k} - 1$

$$\begin{aligned} E(x) &= \frac{1-\frac{k}{n}}{\frac{k}{n}} \\ &= (1-\frac{k}{n}) \cdot \frac{n}{k} \\ &= \frac{n}{k} - 1 \\ E(x) &= \Sigma_{i=0}^{\infty}\Big(1-\frac{k}{n}\Big)^i \cdot \frac{k}{n} \cdot (i+1) \\ &= \Sigma_{i=0}^{\infty}\Big(\frac{n-k}{n}\Big)^i \cdot \frac{k}{n} \cdot (i+1) \end{aligned}$$

Question 2

Written by Huakun Shen; Checked by Jiatao Xiang and Xu Wang
For the sake of writing better-to-understand algorithm, we will make some definitions and assumptions.
We use disjoint set (Forest Structure) with weighted union by size and path compression.
Assume *constraints* and *variables* are stored in 2 lists, where $|constraints| = m, |variables| = n$.
Assume each constaint contains 3 attributes:

- first set (denoted by $constraint[0]$)
- second set (denoted by $constraint[1]$)
- type of constaint (denoted by $constraint[2]$, value is whether *equality* or *inequality*)

Index of these list starts at 1.

```
1  def Algorithm(constraints[], variables[]):
2      for i = 1...n:
3          Si <- variables[i]  # Make n disjoint sets for each element of variables named S1...Sn
4      for constraint in constraints:
5          if constaint[2] is equality and Find(constraint[0]) != Find(constraint[1]):
6              Union(constraint[0], constraint[1])
7      for constraint in constraints:
8          if constaint[2] is inequality:
9              if Find(constraint[0]) == Find(constraint[1]):
10                 return Nil
11     int i = 0
12     for s in sets that are not empty:
13         for element in s:
14             element = i
15         i++
16     return variables
```

Runtime Analysis:

1. Line 2-3 initializes n disjoint sets, each represents a variable.
   It takes $\mathcal{O}(n)$ in total.
2. The loop on line 4 runs m iterations, but since there are n elements, there are at most $(n-1)$ unions.
3. The loop on line 7 runs m iterations, since there are m constraints, there are at most $2m$ *Find*'s (line 9 calls *Find* twice).
4. For the loop on line 12, although it's a nested loop, since there are n elements, and the nested loop only traverse through every one of them, this part takes $\mathcal{O}(n)$ of time.

From lecture, we know that for σ : Sequence of $(n-1)$ *Unions* mixed with $(m \geq n)$ *Finds*, σ takes $\mathcal{O}(m \cdot log^* n)$.
Part 2 and 3 altoghter has at most $(n-1)$ *Unions* and $2m$ *Finds*, which takes $\mathcal{O}(2m \cdot log^* n) = \mathcal{O}(2m \cdot log^* n)$,
Altogether, the algorithm takes $\mathcal{O}(n + m \cdot log^* n + n) = \mathcal{O}(n + m \cdot log^* n) < \mathcal{O}(mn)$.