Solutions for Homework Assignment #4

Answer to Question 1.

a. If the number of *non-empty* slots in T is k before inserting x, what is the probability that x is inserted into an *empty* slot?

Let $S = \{i_1, i_2, \ldots, i_k\}$ be the set of *non-empty* slots of T before inserting x.
The probability that h_1 hashes x into an *non-empty* slot of T is:

$$\begin{aligned} Prob[h_1(x) \in S] &= Prob[h_1(x) = i_1 \vee h_1(x) = i_2 \vee \ldots \vee h_1(x) = i_k] \\ &= \sum_{j=1}^{j=k} Prob[h_1(x) = i_j] \end{aligned}$$

Since T has m slots and we assume that the hash function h_1 satisfies the *Simple Uniform Hashing Assumption* (SUHA), we have $Prob[h_1(x) = i_j] = \frac{1}{m}$ for every j, $1 \leq j \leq k$. Therefore:

$$Prob[h_1(x) \in S] = \sum_{j=1}^{j=k} \frac{1}{m} = \frac{k}{m}$$

Since h_2 also satisfies SUHA, the probability that h_2 hashes x into a *non-empty* slot of T is also:

$$Prob[h_2(x) \in S] = \frac{k}{m}$$

Note that x is inserted in a *non-empty* slot of T if and only if *both* h_1 and h_2 hash into a *non-empty* slot of T. Therefore:

$$Prob[x \text{ is inserted in a } \textit{non-empty} \text{ slot of } T] = Prob[h_1(x) \in S \wedge h_2(x) \in S]$$

Thus, since the hash functions h_1 and h_2 are independent:

$$\begin{aligned} Prob[x \text{ is inserted in a } \textit{non-empty} \text{ slot of } T] &= Prob[h_1(x) \in S] \cdot Prob[h_2(x) \in S] \\ &= \left(\frac{k}{m}\right)^2 \end{aligned}$$

Therefore:

$$\begin{aligned} Prob[x \text{ is inserted in an } \textit{empty} \text{ slot of } T] &= 1 - Prob[x \text{ is inserted in a } \textit{non-empty} \text{ slot of } T] \\ &= 1 - \left(\frac{k}{m}\right)^2 \end{aligned}$$

b. Suppose that $m = 4$, and $T[0]$, contains 4 elements, $T[1]$ contains 2 elements, and $T[2]$, $T[3]$ contain 6 elements each. What is the *expected* length of the chain x is inserted into, *not counting* x itself?

Since the table has 4 slots, when we insert x using the two independent hash functions h_1 and h_2 there are 16 possible hashing outcomes:

$$S = \{(i,j) \mid \text{where } 0 \leq i = h_1(x) \leq 3 \text{ and } 0 \leq j = h_2(x) \leq 3\} \qquad (S \text{ is our sample space})$$

Note that:

$$\begin{aligned} Prob[(i,j)] &= Prob[h_1(x) = i \wedge h_2(x) = j] && \text{(by definition)} \\ &= Prob[h_1(x) = i] \cdot Prob[h_2(x) = j] && \text{(because } h_1 \text{ and } h_2 \text{ are independent)} \\ &= \frac{1}{4} \cdot \frac{1}{4} = \frac{1}{16} && \text{(because } h_1 \text{ and } h_2 \text{ satisfy SUHA)} \end{aligned}$$

Let X be the random variable denoting the length of the chain where x is inserted. That is, for each possible outcome $o = (i, j) \in S$, $X(o)$ is the length of the chain where x is inserted. We are seeking $E[X]$. Note that:

$$\begin{aligned} E[X] &= \sum_{o \in S} X(o) \cdot Prob[o] \\ &= \sum_{(i,j) \in S} X[(i,j)] \cdot Prob[(i,j)] \\ &= \frac{1}{16} \sum_{(i,j) \in S} X[(i,j)] \\ &= \frac{1}{16}\big(X[(0,0)] + X[(0,1)] + X[(0,2)] + X[(0,3)] \\ &+ X[(1,0)] + X[(1,1)] + X[(1,2)] + X[(1,3)] \\ &+ X[(2,0)] + X[(2,1)] + X[(2,2)] + X[(2,3)] \\ &+ X[(3,0)] + X[(3,1)] + X[(3,2)] + X[(3,3)]\big) \end{aligned}$$

Recall that $T[0]$ contains 4 elements, $T[1]$ contains 2 elements, and $T[2]$, $T[3]$ contain 6 elements each. From our hashing scheme, when the hashing outcome is (i, j), the length $X[(i, j)]$ of the chain that x enters is as follows: if $i = 1$ or $j = 1$ then $X[(i, j)] = 2$, else if $i = 0$ or $j = 0$ then $X[(i, j)] = 4$ else $X[(i, j)] = 6$. Therefore:

$$\begin{aligned} E[X] &= \frac{1}{16}\big(4 + 2 + 4 + 4 \\ &+ 2 + 2 + 2 + 2 \\ &+ 4 + 2 + 6 + 6 \\ &+ 4 + 2 + 6 + 6\big) \\ &= \frac{58}{16} \\ &= 3.625 \end{aligned}$$

Another way to compute $E[X]$ is as follows. Note that the chain where x is inserted has length 2, 4 or 6. So the only possible value v of the random variable X is 2, 4 and 6. An alternative formula for $E[X]$ is:

$$E[X] = \sum_{v \in \{2,4,6\}} v \cdot Prob[X = v]$$

It now suffices to compute $Prob[X = 2]$, $Prob[X = 4]$, and $Prob[X = 6]$.

Note that $X = 2$ if and only if the outcome of the hashing is an (i, j) with $i = 1$ or $j = 1$. Therefore:

$$\begin{aligned} Prob[X = 2] &= Prob[\{(0,1),(1,0),(1,1),(1,2),(1,3),(2,1),(3,1)\}] \\ &= Prob[(0,1)] + Prob[(1,0)] + Prob[(1,1)] + Prob[(1,2)] + Prob[(1,3)] + Prob[(2,1)] + Prob[(3,1)]\}] \\ &= 7 \cdot \frac{1}{16} = \frac{7}{16} \end{aligned}$$

Similarly:

$$\begin{aligned} Prob[X = 4] &= Prob[\{(0,0),(0,2),(0,3),(2,0),(3,0)\}] \\ &= Prob[(0,0)] + Prob[(0,2)] + Prob[(0,3)] + Prob[(2,0)] + Prob[(3,0)]\}] \\ &= 5 \cdot \frac{1}{16} = \frac{5}{16} \\ Prob[X = 6] &= Prob[\{(2,2),(2,3),(3,2),(3,3)\}] \\ &= Prob[(2,2)] + Prob[(2,3)] + Prob[(3,2)] + Prob[(3,3)]\}] \\ &= 4 \cdot \frac{1}{16} = \frac{4}{16} \end{aligned}$$

Therefore:

$$\begin{aligned} E[X] &= \sum_{v \in \{2,4,6\}} v \cdot Prob[X = v] \\ &= 2 \cdot \frac{7}{16} + 4 \cdot \frac{5}{16} + 6 \cdot \frac{4}{16} \\ &= \frac{58}{16} \end{aligned}$$

Answer to Question 2.

Let L be input list. L contains a mix of items of the form $S(i, j)$ and $D(k, l)$. let L_S be the ***sublist*** of L that contains only the items of the form $S(i, j)$. For example, if L is:

$$S(5, 1), S(1, 6), S(2, 7), S(4, 5), D(2, 6), S(1, 3), D(3, 4)$$

then L_S is:

$$S(5, 1), S(1, 6), S(2, 7), S(4, 5), S(1, 3)$$

Note that L_S represents a collection $\mathcal{C}_S$ of ***disjoint sets*** of bone labels, where each set contains (the label of) bones that are from the ***same*** specie (by transitivity). In our example, $\mathcal{C}_S$ consists of the two disjoint sets $\{1, 3, 4, 5, 6\}$ and $\{2, 7\}$. So according to the above L_S, the bones come from at most 2 different species.

Recall that $D(k, l)$ states that bone k and l are from ***different*** species. So if the list L contains an item $D(k, l)$ such that both k and l are in the ***same*** set of $\mathcal{C}_S$, then the list L *must be incorrect*. In our example above, L must be incorrect: it contains $D(3, 4)$, but 3 and 4 are in the same set $\{1, 3, 4, 5, 6\}$ of $\mathcal{C}_S$. In other words, while $D(3, 4)$ of L says that bones 3 and 4 are from different species, the set $\{1, 3, 4, 5, 6\}$ derived from the $S(\cdot, \cdot)$ items of L says that 3 and 4 are from the same species!

From the above, the high-level idea of the algorithm is as follows:

1. First use all the $S(\cdot, \cdot)$ items in L to build the collection $\mathcal{C}_S$ of disjoint sets of bone labels that are in the same species.
2. Then for each item $D(k, l)$ in L, determine whether k and l are in the *same* set; if they are, then output ERROR FOUND and stop.
3. Output the number of disjoint sets of bone labels in the collection $\mathcal{C}_S$ that was built in part 1.

One way to implement this algorithm efficiently is to use the disjoint-sets UNION-FIND data structure because with this data structure it is easy to: (1) use the $S(\cdot, \cdot)$ items in L to construct the collection of disjoint sets $\mathcal{C}_S$, and (2) for each item $D(k, l)$ in L determine whether k and l are in the same set of $\mathcal{C}_S$ or not. More precisely:

1. Construct n singleton sets $\{1\}, \{2\}, \ldots, \{n\}$ representing each of the n bone labels $1, 2, \ldots, n$.

   Set $k := n$ (current estimate of the maximum possible number of different species the bones are from).
2. Scan the list L. For each $S(i, j)$ in L, if i and j are in ***different*** sets then:

   (a) ***merge*** these two sets using a UNION operation, and

   (b) decrement k by 1.
3. Re-scan the list L. For each $D(k, l)$ in L, use two FIND operations to determine whether k and l are in the ***same*** set; if they are, then output ERROR FOUND and stop.
4. Output k.

Pseudocode for the algorithm outlined above (where i represents bone labelled i, for $1 \leq i \leq n$):

```
BoneSpecies(n, L)
 1  for i = 1 to n
 2      MakeSet(i)              // element i represents bone labelled i
 3  k = n
 4  for each S(i, j) in L
 5      u = Find(i)
 6      v = Find(j)
 7      if u ≠ v
 8          Union(u, v)
 9          k = k − 1
10  for each D(k, l) in L
11      if Find(k) = Find(l)
12          return Error Found
13  return k
```

Complexity Analysis. For efficiency it is best to use the ***forest implementation*** of the disjoint-sets data structure with the weighted union (WU) and path compression (PC) heuristics. Recall that $m \geq n$. Note that the above algorithm performs $O(n)$ MakeSets, $O(n)$ Unions, and $O(m)$ Finds. Thus, the worst-case time complexity of this algorithm is $O(m \log^* n)$.