CS302 - Final Review

1. **Dynamic Programming** Given two strings $x = x_1x_2x_2 \dots x_n$ and $y = y_1y_2y_2 \dots y_m$, the edit distance $D(x, y)$ is the minimum number of insertions or deletions or substitutions (a substitution involves replacing one character in the string with another) required to turn x into y (or vice versa). This is used for spellcheckers: if someone types a word that is not in the dictionary, you want to find the word that is closest to it in edit distance. For example, if someone typed "graffe" its edit distance from "graft" is 2 (delete "e", substitute "f" for "t"), while its edit distance to "giraffe" is 1 (insert "i").

   We want to design a dynamic programming algorithm to determine the shortest edit distance.

   (a) If we consider the optimal way of turning x into y, what options do we have for how to make that transition? (Think about the e.g. smallest/first/last choices we could make.)

   (b) Let $\vec{x}_i$ be the first i letters in the string x and $\vec{y}_i$ be the first i letters in the string y. Explain how $D(x, y) = D(\vec{x}_n, \vec{y}_m)$ is related to the edit distance of substrings in each option from the previous part.

   (c) Combine the results into one recurrence relation. What is the base case?

   (d) Let A be the array that you creat in your algorithm. What should you store in the array? What should the size of the array be?

   (e) Which rows should you fill out first?

   (f) How long will it take to fill in the rest of A.

   (g) Write pseudocode that, if given a properly filled out A as input, tells you how to turn x into y.

   (h) Prove using a loop invariant that your pseudocode is correct. (You may assume that A is properly filled out, and that you previously proved the correctness of your recurrence relation.)

Solution

(a) We can (i) delete the last element of x, we can (ii) add a new last element to x (which should be the same as the last element of y), we can (iii) substitute the last element of x for a different element, or we can (iv) leave the last element of x alone.

(b) (i)

$$D(\vec{x}_n, \vec{y}_m) = 1 + D(\vec{x}_{n-1}, \vec{y}_m) \tag{1}$$

(ii)

$$D(\vec{x}_n, \vec{y}_m) = 1 + D(\vec{x}_n, \vec{y}_{m-1}) \tag{2}$$

(iii)

$$D(\vec{x}_n, \vec{y}_m) = 1 + D(\vec{x}_{n-1}, \vec{y}_{m-1}) \tag{3}$$

(iv)

$$D(\vec{x}_n, \vec{y}_m) = D(\vec{x}_{n-1}, \vec{y}_{m-1}) \tag{4}$$

(c)

$$D(\vec{x}_n, \vec{y}_m) = \min\{1 + D(\vec{x}_{n-1}, \vec{y}_m), 1 + D(\vec{x}_n, \vec{y}_{m-1}), 1 + D(\vec{x}_{n-1}, \vec{y}_{m-1}), D(\vec{x}_{n-1}, \vec{y}_{m-1})\} \tag{5}$$

where the final option is only allowed if the last letter in x is the same as the last letter in y.

$$D(\emptyset, \vec{y}_i) = i \qquad D(\vec{x}_i, \emptyset) = i \qquad D(\emptyset, \emptyset) = 0 \tag{6}$$

(d) A should be $n + 1 \times m + 1$ in size, and $A[i, j]$ should store $D(\vec{x}_i, \vec{y}_j)$.

(e) Fill out the row $A[0, :]$ and the column $A[:, 0]$ first.

(f) It will take $O(nm)$ time to fill in the array.

(g) I ran out of time. Please come discuss your answer with me!

2. **NP** A variant of Subset-Sum involves a set that contains both positive and negative integers, and the target value is always 0. Call this variant Subset-Sum-0. Prove that Subset-Sum-0 is NP-Complete

Solution Subset-Sum-0 is in NP because if the input contains n integers of less than m digits, the solution would be some subset of them, so it would require $O(nm)$ space to write down the solution. Then we can add n integers of m digits together in $O(nm)$ time to check if they sum to 0.

To show Subset-Sum-O is NP Hard, we show 3-SAT reduces to Subset-Sum-0. To do this, we use exactly the same reduction as before, except we include the integer -11111111133333333333 in our set. All of the other numbers are positive, so the only way to get to 0 is if we also can get to 11111111133333333333, and we previously showed this occurs iff there is a solution to 3-SAT.

3. **Randomized Algorithms** What is the average runtime of randomized search without replacement if there are c copies of the item we are looking for out of an array of size n. Please use indicator random variables and go through the usual routine. You do not need to simplify your final answer - it can be a messy sum.

Solution The sample space is the set of all possible choices of guessed elements. Let X be the random variable that is the number of guesses required to find an item we are looking for. Let X_i be the indicator random variable that takes value 1 if we have at least i rounds. Then

$$X = \sum_{i=1}^{n-c+1} X_i \tag{7}$$

so using linearity of expectation

$$\mathbb{E}[X] = \sum_{i=1}^{n-c+1} \mathbb{E}[X_i] \tag{8}$$

and using the properties of indicator random variables:

$$\mathbb{E}[X] = \sum_{i=1}^{n-c+1} Pr(\text{at least } i \text{ rounds occur}) \tag{9}$$

Now the probability that there are at least i rounds is the probability that we haven't found one of the items in the first $i-1$ rounds. This is

$$\frac{n-c}{n} \times \frac{n-c-1}{n-1} \times \frac{n-c-2}{n-2} \times \cdots \times \frac{n-c-(i-2)}{n-(i-2)} \tag{10}$$

Plugging in

$$\mathbb{E}[X] = 1 + \sum_{i=2}^{n-c+1} \frac{n-c}{n} \times \frac{n-c-1}{n-1} \times \frac{n-c-2}{n-2} \times \cdots \times \frac{n-c-(i-2)}{n-(i-2)} \tag{11}$$