Dynamic Programming: Edit Distance[1]

In this lecture, we will look at another class of poster-child problems for dynamic programming: "string" problems. The input to these problems will be strings of the form $s[1:m]$ where each $s[i]$ will be from some alphabet Σ. This alphabet could be $\{0,1\}$, the alphabet of ASCII symbols, or $\{A,C,G,T\}$, depending on the applications.

How far is the string `algorithm` from the string `logarithm`? How far is `apple` from `banana`? How far is the cat-genome from the mouse-genome? Does it make sense to ask these questions? Well, the third question may point one to the importance of such a question. But how do we make the question well-defined so that we can to talk about it? The *edit distance* or the *Levenshtein distance* is one way to capture this.

Given two strings $s[1:m]$ and $t[1:n]$, the edit distance captures the notion of distance between s and t defined using 3 operations. The first is the *insert* operation, $\mathsf{ins}(s,i,c)$, which inserts character c after $s[i]$, thus making s longer; $\mathsf{del}(s,j)$ deletes $s[j]$ from s making it shorter; and $\mathsf{sub}(s,i,c)$ replaces $s[i]$ with the character c keeping the length the same. Each operation is assumed to cost 1 unit. The edit distance between $s[1:m]$ and $t[1:n]$ is the minimum cost sequence of operations of the kind ins, sub, del which can required to convert s into t. This cost is called the *edit distance* between s and t, and is denoted as $\mathsf{ED}(s,t)$.

For example, if s is `apple` and t is `banana`, then $\mathsf{ED}(s,t) \leq 5$ since one can go from `apple` $\to$ `bapple` $\to$ `banple` $\to$ `banale` $\to$ `banane` $\to$ `banana`. The operations are $\mathsf{ins}(s,0,b)$, $\mathsf{sub}(s,3,n)$, $\mathsf{sub}(s,4,a)$, $\mathsf{sub}(s,5,n)$, and $\mathsf{sub}(s,6,a)$. Turns out, this is the best one can do for these two strings. Finding the edit distance is the problem we will look at today.

EDIT DISTANCE
Input: Two strings $s[1:m]$ and $t[1:n]$.
Output: Return $\mathsf{ED}(s,t)$.
Size: m, n.

At first glance, the problem seems to be rather complicated. For instance, suppose we had to find the edit distance between `apple` and `rallies`. One sees that the sequence of letters (`a`, `l`, `e`) appears in both strings in that order, although not contiguously. It seems like a good idea (not saying the best idea) to not touch them, and then move using insertions and substitutions as `apple` $\to$ `rapple` $\to$ `raplle` $\to$ `raplles` $\to$ `ralles` $\to$ `rallies`.

However, the number of such subsequences can be pretty large. Indeed, it corresponds to a *subset* of locations on the strings, and the number of subsets (as in the subset-sum and knapsack) is *exponentially large* (it's 2^n). Going over all of them is a bad idea. Again, dynamic programming will come to our rescue.

As in SUBSET SUM and KNAPSACK, we imagine the "best" solution which takes us from s to t. Note that a solution for an EDIT DISTANCE instance, is a sequence π of operations where each entry is an ins, del, or sub. The cost of the solution, $\mathsf{ED}(s,t)$ is simply the length $|\pi|$ of this sequence. Let π^* be the best sequence (which we don't know), but we want to argue about its structure. In particular, we want to argue this solution contains solutionettes.

To do so, let us focus on the *last* entries of both strings s and t. For example, if s = `apple` and t = `rallies`. The last entries are $s[m]$ = `e` and $t[n]$ = `s`. After we perform the operations in π^*, the character $s[m]$ at the end must become the character $t[n]$ at the end. There are *four* ways this can occur.

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 19th Mar, 2022*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

- One, $t[n]$ was introduced using a ins operation somewhere in π^*. This could occur, for instance, if $s =$ pan and $t =$ any where π^* is pan $\to$ pany $\to$ any. The character $t[n] =$ y is inserted in the first step of π^*.
- Two, maybe the character $t[n]$ was already present in the string s. In that case, we get there by using a del on $s[m]$ somewhere in π^*. This could occur, for instance, if $s =$ ate and $t =$ cat where π^* is ate $\to$ cate $\to$ cat. The character $t[n] =$ t was already present, and the character $s[m] =$ e was deleted in the second step of π^*.
- Three, maybe the $t[n]$ *substituted* the last entry $s[m]$. This could occur, for instance, if $s =$ pan and $t =$ pit where π^* is pan $\to$ pat $\to$ pit. The character $t[n] =$ t substitutes $s[m] =$ n in the first step of π^*.
- Four, maybe $s[m]$ and $t[n]$ are the *same* characters, in which case we can leave them alone. This could occur, for instance, if $s =$ bag and $t =$ cog where π^* is bag $\to$ bog $\to$ cog.

To make dynamic programming work, we need to see if the *remaining* steps of π^* give an *optimum solution* for a smaller sub-instance of EDIT DISTANCE. If so, then we will have discovered the recursive substructure. Indeed, in all the four cases described above, we do have this.

- In this case, the remaining operations of π^* takes $s[1 : n]$ to $t[1 : m - 1]$. In the example above, it would be pan $\to$ an.
- In this case, the remaining operations of π^* takes $s[1 : n - 1]$ to $t[1 : m]$. In the example above, it would be at $\to$ cat.
- In this case, the remaining operations of π^* takes $s[1 : n - 1]$ to $t[1 : m - 1]$. In the example above, it would be pa $\to$ pi.
- In this case, not the remaining but the *whole* of π^* takes $s[1 : n - 1]$ to $t[1 : m - 1]$. In the example above, it would be ba $\to$ co.

Therefore, from the instance $I = (s[1 : m], t[1 : n])$, we get three smaller subinstances $I_1 = (s[1 : m - 1], t[1 : n - 1])$, $I_2 = (s[1 : m], t[1 : n - 1])$, and $I_3 = (s[1 : m - 1], t[1 : n])$, and it seems that (a) a solution of I leads to a solution of I_1 or I_2 or I_3, and (b) solutions of I_1, I_2, I_3 lead to solutions of I (once again, formal proof comes later – first the idea). If one were to draw the recursion tree more, one would observe that a "typical subinstance' would look like like $I' = (s[1 : i], t[1 : j])$. Thus, these are parameterized by $0 \leq i \leq m$ and $0 \leq j \leq n$, and therefore can be arranged in an $(m + 1) \times (n + 1)$ grid. The base case: when $i = 0$ or $j = 0$, that is when one of the strings is empty, in which case the edit distance is the length of the other string. We have all the ingredients for the dynamic programming solution which we now rigorously provide below in our usual six-step procedure.

a. ***Definition:*** For any $0 \le i \le m$ and $0 \le j \le n$, let us use $\mathsf{ED}(i,j)$ to be the edit distance between the strings $s[1:i]$ and $t[1:j]$. We are interested in $\mathsf{ED}(m,n)$.

   What should $\mathsf{Cand}(i,j)$ be? Since the edit distance is the smallest number of "string operations" (ins/del/sub), let's define $\mathsf{Cand}(i,j)$ as the all possible sequences π of string operations which take the string $s[1:i]$ to the string $t[1:j]$. Armed with this notation, we get

$$\mathsf{ED}(i,j) = \min_{\pi \in \mathsf{Cand}(i,j)} |\pi|$$

b. ***Base Cases:***

   $\mathsf{ED}(0,j) = j$ for all $0 \le j \le n$ and $\mathsf{ED}(i,0) = i$ for all $0 \le i \le m$. There is only one way to go from an empty string to a string j – keep inserting. There is only one way to from a string of length i to an empty string – keep deleting.

c. ***Recursive Formulation:*** Since we need to know whether the last characters of the strings are equal or not, let us introduce a piece of notation which will help us. Let $\mathbf{1}_{i,j}$ be the indicator variable of whether $s[i] = t[j]$ formally defined as

$$\mathbf{1}_{i,j} = \begin{cases} 1 & \text{if } s[i] = t[j] \\ 0 & \text{otherwise} \end{cases}$$

   For all $i > 0, j > 0$:

$$\mathsf{ED}(i,j) = \min(\ 1 + \mathsf{ED}(i-1,j),\ 1 + \mathsf{ED}(i,j-1),\ (1 - \mathbf{1}_{i,j}) + \mathsf{ED}(i-1,j-1)\ )$$

d. ***Formal Proof:***

   ($\ge$): Let π^* be the sequence of operations which took $s[1:i]$ to $t[1:j]$ and $|\pi^*| = \mathsf{ED}(i,j)$. Note that, in π^*, either $s[i]$ is deleted from the end, or $t[j]$ is inserted at the end, and if neither of these two occur, we must either substitute $s[i]$ and $t[j]$, or these characters are the same. In the first case, consider the sequence of operations π which is π^* without the deletion. Observe, that π acting on $s[1:i-1]$ would take us to $t[1:j]$. Thus, $|\pi^*| = 1 + |\pi| \ge 1 + \mathsf{ED}(i-1,j)$. Similarly, in the second case, consider the sequence π which is π^* without the insertion. π takes us from $s[1:i]$ to $t[1:j-1]$, and thus, in this case, $|\pi^*| \ge 1 + \mathsf{ED}(i,j-1)$. Finally, if neither of the above two occur, then either $s[i] = t[j]$ in which case π^* actually takes $s[1:i-1]$ to $t[1:j-1]$. That is, $\pi^* \ge \mathsf{ED}(i-1,j-1) = (1-\mathbf{1}_{i,j}) + \mathsf{ED}(i-1,j-1)$ since $s[i] = t[j]$. Or, $s[i] \ne t[j]$, and there is a substitution. And in this case, π defined as π^* minus that substitution takes $s[1:i-1]$ to $t[1:j-1]$. Again giving, $\pi^* \ge 1 + \mathsf{ED}(i-1,j-1) = (1 - \mathbf{1}_{i,j}) + \mathsf{ED}(i-1,j-1)$ in this case. In sum, in all of the possible cases, $\mathsf{ED}(i,j) = |\pi^*|$ is larger than one of the things in the RHS parenthesis.

   ($\le$): Let π be the sequence of operations in $\mathsf{Cand}(i-1,j)$ of length $\mathsf{ED}(i-1,j)$. Consider the sequence of operations $\pi' = \mathsf{del}(s,i) \circ \pi$, which first *deletes* the last entry of $s[1:i]$ to get $s[1:i-1]$, and then follows the sequence of operations in π to get to $s[1:j]$. Then, π' takes $s[1:i]$ to $t[1:j]$ and thus, $\pi' \subseteq \mathsf{Cand}(i,j)$. Therefore, $|\pi'| \ge \mathsf{ED}(i,j)$. Since $|\pi'| = 1 + |\pi| = 1 + \mathsf{ED}(i,j)$, we get that $\mathsf{ED}(i,j) \le 1 + \mathsf{ED}(i-1,j)$. Similarly, one can show

$\mathsf{ED}(i,j) \leq 1 + \mathsf{ED}(i,j-1)$; the only difference is that we would $\mathsf{ins}(t[1:j-1], t[j], j)$ at the end of doing π.

Finally, suppose π was a sequence of operations that took $s[1:i-1]$ to $t[1:j-1]$ and whose length was $\mathsf{ED}(i-1,j-1)$. If $s[i] = t[j]$, and thus $\mathbf{1}_{i,j} = 1$, then π also takes $s[1:i]$ to $t[1:j]$. And so, $|\pi| \geq \mathsf{ED}(i,j)$ implying $\mathsf{ED}(i,j) \leq (1-\mathbf{1}_{i,j}) + \mathsf{ED}(i-1,j-1)$. If $s[i] \neq t[j]$, and thus $\mathbf{1}_{i,j} = 0$, then consider the sequence $\pi' = \mathsf{sub}(s, t[j], i) \circ \pi$. π' takes $s[1:i]$ to $t[1:j]$ and thus $|\pi'| \geq \mathsf{ED}(i,j)$. Since $|\pi'| = 1 + |\pi| = (1 - \mathbf{1}_{i,j}) + \mathsf{ED}(i-1,j-1)$, we get $\mathsf{ED}(i,j) \leq (1 - \mathbf{1}_{i,j}) + \mathsf{ED}(i-1,j-1)$.

e. ***Pseudocode for computing*** $\mathsf{ED}(m,n)$**.**

```
procedure ED($s[1:m], t[1:n]$):
    ▷ Returns the edit distance between $s$ and $t$.
    Allocate space $E[0:m, 0:n]$ ▷ $E[i,j]$ will contain the edit distance between $s[1:i]$ and $t[1:j]$.
    $E[0,j] \leftarrow j$ for all $0 \le j \le n$ and $E[i,0] \leftarrow i$ for all $0 \le i \le m$. ▷ Base Cases.
    for $1 \le i \le m$ do:
        for $1 \le j \le n$ do:
            if $s[i] = t[j]$ then:
                $\mathbf{1}_{i,j} \leftarrow 1$
            else:
                $\mathbf{1}_{i,j} \leftarrow 0$
            $E[i,j] \leftarrow \min(\ E[i-1,j]+1,\ E[i,j-1]+1,\ E[i-1,j-1]+(1-\mathbf{1}_{i,j})\ )$
    return $E[m,n]$.
    ▷ $E[m,n]$ now contains the value of the edit distance
    ▷ Below we show the "recovery" pseudocode which shows one way to get from $s$ to $t$ in $E[m,n]$ moves.

    $i \leftarrow m; j \leftarrow n; \pi = []$
    ▷ Invariant: $|\pi| + E[i,j] = E[m,n]$
    while $i > 0$ and $j > 0$ do:
        if $E[i,j] = E[i-1,j-1] + (1-\mathbf{1}_{i,j})$ then:
            if $\mathbf{1}_{i,j} \ne 1$ that if $s[i] \ne t[j]$ then: ▷ Substititure
                Append sub$(s, i, t[j])$ to $\pi$
            $i \leftarrow i-1; j \leftarrow j-1$
        else if $E[i,j] = 1 + E[i-1,j]$ then:
            ▷ We must have deleted $s[i]$
            Append del$(s, i, s[i])$ to $\pi$
            $i \leftarrow i-1$
        else: ▷ We must have that $E[i,j] = 1 + E[i,j-1]$ and we must have inserted $t[j]$ at the end
            Append ins$(s, i, t[j])$ to $\pi$
            $j \leftarrow j-1$
    ▷ At this point either $i = 0$ or $j = 0$. Depending on which, we need to do a few more operations.
    while $i > 0$ do:
        Append del$(s, i, s[i])$ to $\pi$
        $i \leftarrow i-1$
    while $j > 0$ do:
        Append ins$(s, i, t[j])$ to $\pi$
        $j \leftarrow j-1$
    return Reverse of $\pi$, $E[m,n]$
```

f. ***Running time and space*** The above pseudocode take $O(mn)$ time and space.

Theorem 1. The EDIT DISTANCE between two strings can be found in $O(nm)$ time and space.