Week 04: Alignment

FanFly

March 29, 2020

Edit Distance

Edit distance is a way to quantify the "distance" between two strings.

- The distance is calculated by counting the minimum number of operations needed to transform one string into the other.
- Different definitions of an edit distance use different sets of string operations.

Edit Distance (cont.)

Suppose that we want to find the edit distance between `weather` and `whether`, and we only allow insertion and deletion of a character in a string.

- We can transform `weather` into `whether` by inserting an `h` and deleting an `a`, and thus the edit distance between them is 2.
- Note that if we remove the different parts from both strings, we will get their **longest common subsequence**, i.e., `wether`.
- Conversely, if we can find their longest common subsequence, we can calculate their edit distance.

Longest Common Subsequence

Now we introduce the longest common subsequence problem.

The Longest Common Subsequence Problem

- Input: Two sequences A and B of lengths n and m, respectively.
- Output: A longest common subsequence of A and B.

We can find a longest common subsequence of two sequences by finding an alignment between them.

```
w-eather
whe-ther
```

Note that there may be more than one longest common subsequences.

```
mea--surement     -measur-ement
--amus--ement     am---u-sement
```

Recurrence Relation

Let us look at some examples.

```
c-a-ke   -m--ask   googol-
-fad-e   i-deas-   goog-le
```

Let $A = A' \cdot x$ and $B = B' \cdot y$, where x and y are the last elements of A and B, respectively.

- If $x = y$, then we have

$$\mathrm{LCS}(A, B) = \mathrm{LCS}(A', B') \cdot x.$$

- Otherwise, we have

$$\mathrm{LCS}(A, B) = \mathrm{LCS}(A, B') \quad \text{or} \quad \mathrm{LCS}(A, B) = \mathrm{LCS}(A', B).$$

Recursive Method

Thus, it can be solved recursively!

```
def lcs(A, B, n, m):
    if n == 0 or m == 0:
        return A[:0]
    elif A[n - 1] == B[m - 1]:
        return lcs(A, B, n - 1, m - 1) + A[n - 1]
    else:
        P = lcs(A, B, n - 1, m)
        Q = lcs(A, B, n, m - 1)
        if len(P) > len(Q):
            return P
        else:
            return Q
```

Time Complexity of Recursive Method

What is its time complexity? We have

$$T(n,m) = \begin{cases} O(1), & \text{if } n = 0 \text{ or } m = 0 \\ T(n,m-1) + T(n-1,m) + O(1), & \text{otherwise.} \end{cases}$$

Thus, we can conclude that

$$T(n,m) = \Theta\left(\frac{(n+m)!}{n! \cdot m!}\right),$$

which is really inefficient.

Dynamic Programming

Why the recursive method is so inefficient?

- It solves the same subproblem over and over again.
- If the algorithm can remember the solutions to the solved subproblems, then it can become more efficient.

	m	a	s	k
i	-	-	-	-
d	-	-	-	-
e	-	-	-	-
a	-	a	a	a
s	-	a	as	as

- This improvement is called **dynamic programming**.

Dynamic Programming (cont.)

With dynamic programming, we can solve the longest common subsequence problem efficiently.

- Assume that the length of longest common subsequence is ℓ.
- The content of each entry can be computed in $\Theta(\ell)$ time.
- Thus, we can solve the problem in $O(nm\ell)$ time.

Improvement

In fact, the time complexity can be improved to $O(nm)$ if we only memoize the length of the longest common subsequence.

	m	a	s	k
i	0	0	0	0
d	0	0	0	0
e	0	0	0	0
a	0	1	1	1
s	0	1	2	2

Exercise #1

Let A and B be sequences of lengths n and m, respectively.

If the length of longest common subsequence of A and B is ℓ, what is the edit distance between A and B?

- Assume that only insertion and deletion are allowed to perform.

Solution

The edit distance between A and B is $(n - \ell) + (m - \ell) = n + m - 2\ell$.

Exercise #2

Let A be a sequence of length n that does not have repeating elements. Hoe many nonempty subsequences does A have?

Solution

It has $2^n - 1$ nonempty subsequences.

Exercise #3

Let A and B be sequences and let f be a **scoring function**. Suppose that

$$f(x, y) = \begin{cases} p, & \text{if } x = y \\ q, & \text{otherwise.} \end{cases}$$

For any alignment, we can compute its score according to the scoring function.

Exercise #3 (cont.)

For example, the alignment below has score $3p + 6q$.

```
goo---gle
googol---
```

Please determine the value of p and q such that the maximum score of alignment of A and B is equal to the length of their longest common subsequence.

Solution

This property is satisfied when $p = 1$ and $q = 0$.