

Lecture 19: Edit Distance

Agenda:

- LCS
- Edit distance

Reading:

- Text: (partly covered) 350-355

Lecture 19: Dynamic Programming

Longest common subsequence (LCS) problem:

- Definitions:
- Sequence/string:
dynamicprogramming is a sequence over the English alphabet
 - Base/letter/character
 - Subsequence:
the given sequence with zero or more letters deleted
e.g., dog is a subsequence of dynamicprogramming
WARNing: bases appear in the same order, but not necessarily consecutive
 - Common subsequence
 - LCS problem: given two sequences $X = x_1x_2 \dots x_n$ and $Y = y_1y_2 \dots y_m$, find a maximum-length common subsequence of them.
- The LCS problem has the “optimal substructure” ...
 - if x_n is NOT in the LCS (to be computed), then we only need to compute an LCS of $x_1x_2 \dots x_{n-1}$ and $y_1y_2 \dots y_m$...
 - similarly, if y_m is NOT in the LCS (to be computed), then we only need to compute an LCS of $x_1x_2 \dots x_n$ and $y_1y_2 \dots y_{m-1}$...
 - if x_n and y_m are both in the LCS (to be computed), then $x_n = y_m$ and we need to compute an LCS of $x_1x_2 \dots x_{n-1}$ and $y_1y_2 \dots y_{m-1}$;
and then adding x_n to the end to form an LCS for the original problem

Lecture 19: Dynamic Programming

Longest common subsequence (LCS) problem (cont'd):

- Therefore,

Letting $DP[n, m]$ to denote the length of an LCS of X and Y , then it is equal to

$$\text{max length of } \begin{cases} LCS(x_1x_2 \dots x_{n-1}, y_1y_2 \dots y_m), \\ LCS(x_1x_2 \dots x_n, y_1y_2 \dots y_{m-1}), \\ LCS(x_1x_2 \dots x_{n-1}, y_1y_2 \dots y_{m-1}) + 1, \end{cases} \text{ if } x_n = y_m$$

- Correctness
- In general, let $DP[i, j]$ denote the length of an LCS of $x_1x_2 \dots x_i$ and $y_1y_2 \dots y_j$.
- Recurrence:

If $i = 0$ or $j = 0$ then clearly $DP[i, j] = 0$.

Else

$$DP[i, j] = \max \begin{cases} DP[i-1, j], \\ DP[i, j-1], \\ DP[i-1, j-1] + 1, \end{cases} \text{ if } x_i = y_j$$

Lecture 19: Dynamic Programming

Longest common subsequence (LCS) problem (cont'd)

— solving the recurrence:

- Pseudocode

```
procedure dpLCS( $X, Y$ )  
  
   $n \leftarrow \text{length}[X]$   
   $m \leftarrow \text{length}[Y]$   
  for  $i \leftarrow 1$  to  $m$  do  
     $DP[i, 0] \leftarrow 0$   
  for  $j \leftarrow 0$  to  $n$  do  
     $DP[0, j] \leftarrow 0$   
  for  $i \leftarrow 1$  to  $n$  do  
    for  $j \leftarrow 1$  to  $m$  do  
      if  $x_i = y_j$  then  
         $DP[i, j] \leftarrow DP[i - 1, j - 1] + 1$   
      else if  $DP[i - 1, j] \geq DP[i, j - 1]$  then  
         $DP[i, j] \leftarrow DP[i - 1, j]$   
      else  
         $DP[i, j] \leftarrow DP[i, j - 1]$   
  return  $DP[n, m]$ 
```

- Running time: $\Theta(n \times m)$

There are $n \times m$ entries each takes constant time to compute.

- Step 4: to find the LCS, store extra information and trace back!

Editing Sequences:

- Given two sequences S and T over alphabets $\Sigma = \{a, b, c\}$.
- Starting with S , at each step we can either:
 - delete a letter from S ,
 - insert a letter from Σ into S (at any place),
 - exchange a letter of S with another one in Σ .
- each of these operations has a cost (given).
- If we use δ to denote “null” then:
 - cost of exchanging a with b is: $Cost[a, b]$, and
 - cost of deleting c (i.e. replacing it with δ) is $Cost[c, \delta]$, and
 - cost of inserting letter b (i.e. replacing δ with b) is $Cost[\delta, b]$.
- Given S , T , and cost table $Cost$, want to find a sequence of operations that transform S into T with minimum cost.
- We assume that $Cost$ is metric:
 - $Cost[\alpha, \alpha] = 0$ for $\alpha \in \Sigma$,
 - $Cost[\alpha, \beta] = Cost[\beta, \alpha]$, for $\alpha, \beta \in \Sigma \cup \{\delta\}$,
 - $Cost[\alpha, \beta] \leq Cost[\alpha, \sigma] + Cost[\sigma, \beta]$, for $\alpha, \beta, \sigma \in \Sigma \cup \{\delta\}$.

Lecture 19: Edit Distance

Edit Distance:

- Let's assume $S = s_1 s_2 \dots s_n$ and $T = t_1 t_2 \dots t_m$.
- Subproblems will be: best edit distance of $s_1 \dots s_i$ and $t_1 \dots t_j$ with $1 \leq i \leq n$ and $1 \leq j \leq m$.
- Let $Edit[i, j]$ denote the minimum cost of editing $s_1 \dots s_i$ into $t_1 \dots t_j$.
- In the optimal solution, either:
 - s_i is deleted and the rest of $s_1 \dots s_{i-1}$ is transformed into $t_1 \dots t_j$, or
 - $s_1 \dots s_i$ is transformed into $t_1 \dots t_{j-1}$ and we insert t_j at the end, or
 - s_i is changed into t_j and the rest of $s_1 \dots s_{i-1}$ is transformed into $t_1 \dots t_{j-1}$.
- Thus:

$$Edit[i, j] = \min \begin{cases} Edit[i-1, j] + Cost[s_i, \delta], \\ Edit[i, j-1] + Cost[\delta, t_j], \\ Edit[i-1, j-1] + Cost[s_i, t_j], \end{cases}$$

- with $Edit[i, 0] = \sum_{k=1}^i Cost[s_k, \delta]$ and $Edit[0, j] = \sum_{k=1}^j Cost[\delta, t_k]$.

Edit Distance:

- Pseudocode

```

procedure Edit-Dist( $S, T, Cost$ )

 $n \leftarrow length[S]$ 
 $m \leftarrow length[T]$ 
 $Edit[0, 0] \leftarrow 0$ 
for  $i \leftarrow 1$  to  $n$  do
     $Edit[i, 0] \leftarrow Edit[i - 1, 0] + Cost[S[i], \delta]$ 
for  $j \leftarrow 1$  to  $m$  do
     $Edit[0, j] \leftarrow Edit[0, j - 1] + Cost[\delta, T[j]]$ 
for  $i \leftarrow 1$  to  $n$  do
    for  $j \leftarrow 1$  to  $m$  do
         $Op1 \leftarrow Edit[i - 1, j] + Cost[S[i], \delta]$ 
         $Op2 \leftarrow Edit[i, j - 1] + Cost[\delta, T[j]]$ 
         $Op3 \leftarrow Edit[i - 1, j - 1] + Cost[S[i], T[j]]$ 
         $Edit[i, j] = \min\{Op1, Op2, Op3\}$ 
return  $Edit[n, m]$ 
    
```

- Running time: $\Theta(n \times m)$

There are $n \times m$ entries each takes constant time to compute.

- To find the sequence of operations: keep extra information and trace back!

Lecture 19: Edit Distance

Have you understood the lecture contents?

well	ok	not-at-all	topic
☐	☐	☐	LCS
☐	☐	☐	edit distance
☐	☐	☐	DP for edit distance