

ALGORITHM DESIGN TECHNIQUES

Dynamic Programming : String / Text Problems: Examples

- Longest Common Subsequence

VERSIONING SYSTEMS

- Consider a collaborative project:
 - Documents (e.g. program files) are edited by more than one person at more than one time.
 - Often one may want to go back to a previous version or one specific person's version.
 - It would be expensive to store copies of every version – particularly so, if changes between versions are small compared to the full text
 - Store only the differences!
 - e.g.
 - Versioning systems in Unix (e.g. *cv*s)

VERSIONING AND DIFFERENCE

- Versioning systems in Unix (e.g. **cvs**) store only the differences between versions.
 - i.e. for every new version created, the difference between the previous version and this is computed.
- Exercise:
 - Read the **man** pages and use the **diff** command on Unix / Linux to understand exactly what is computed.
- Typically these tools define the difference as the **longest common subsequence** between two sequences
 - where each sequence is a (text) file
 - and each term – in the sequence – is a line of text.

DEFINITION: SUBSEQUENCE

- Given a sequence of terms $[T_1, T_2, \dots, T_n]$
 - a subsequence is a sequence of the form $[T_{i_1}, T_{i_2}, \dots, T_{i_m}]$
 - where $1 \leq i_1 < i_2 < \dots < i_m \leq n$
- i.e. a subsequence consists of terms of the sequence – not necessarily contiguous terms – but in the same order.

DIFFERENCE AND COMMON SUBSEQUENCE

- Tools such as *diff* and others define difference between the source sequence (**ss**) and target sequence (**ts**) as:
 - $\text{diff}(\text{ts}, \text{ss}) \cong \text{edits}(\text{ts}, \text{ss})$
 - $\quad \quad \quad \equiv \text{deletions}(\text{css}, \text{ss}) + \text{additions}(\text{ts}, \text{css})$
- where
 - **css** is the common sub-sequence of **ss** and **ts**.
- Since one would try to compute the minimal edits required it is appropriate to compute the maximal common subsequence
 - *i.e. the longest common subsequence if all (kinds of) edits cost the same.*

PROBLEM: LONGEST COMMON SUBSEQUENCE (LCS)

- We denote a sequence as $S[1..n]$ where n is the number of terms in the sequence.
- Let $X[1..m]$ and $Y[1..n]$ be the given sequences and $Z[1..k]$ be any LCS of X and Y .
 - We can define LCS by asking whether the last terms of the sequences match:
 - Is $X[m] = Y[n]$?
 - i.e. we have two cases to consider
 - case $X[m] = Y[n]$
 - case $X[m] \neq Y[n]$

and we need to define Z for each case.

PROBLEM LCS – ANALYSIS

- Let $X[1..m]$ and $Y[1..n]$ be the sequences and $Z[1..k]$ be any LCS of X and Y .
- Then Z can be defined as follows:
 - case $X[m] = Y[n]$
 - $Z[k] = X[m]$ and
 - $Z[1..k-1]$ is the LCS of $X[1..m-1]$ and $Y[1..n-1]$
 - case $X[m] \neq Y[n]$
 - There are two possibilities to consider:
 - Exclude $X[m]$ OR
 - Exclude $Y[n]$
 - whichever results in a longer LCS
- Question:
 - Why are these two possibilities sound (i.e. meaningful) and complete (i.e. exhaustive)?

PROBLEM LCS – ANALYSIS

[2]

- Let $X[1..m]$ and $Y[1..n]$ be the sequences and $Z[1..k]$ be any LCS of X and Y . Then Z can be defined as follows:
 - case $X[m] = Y[n]$
 - $Z[k] = X[m]$ and
 - $Z[1..k-1]$ is the LCS of $X[1..m-1]$ and $Y[1..n-1]$
 - case $X[m] \neq Y[n]$
 - max of
 - case Exclude $X[m]$
 - $Z[1..k]$ is an LCS of $X[1..m-1]$ and $Y[1..n]$
 - case Exclude $Y[n]$
 - $Z[1..k]$ is an LCS of $X[1..m]$ and $Y[1..n-1]$
- Note:
 - This essentially concludes that optimal sub-structure property holds for LCS

PROBLEM LCS: RECURRENCE RELATION

- We can now formulate a recurrence for the length of the LCS of $X[1..i]$ and $Y[1..j]$ as:

$L[i,j] =$

$$\begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ 1 + L[i-1,j-1] & \text{if } i>0, j>0, \text{ and } X[i] = Y[j] \\ \max(L[i,j-1], L[i-1,j]) & \text{if } i>0, j>0 \text{ and } X[i] \neq Y[j] \end{cases}$$

PROBLEM LCS: RECURRENCE RELATION

- We can now formulate a recurrence for the length of the LCS of $X[1..i]$ and $Y[1..j]$ as:

$L[i,j] =$

$$\begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ 1 + L[i-1,j-1] & \text{if } i>0, j>0, \text{ and } X[i] = Y[j] \\ \max(L[i,j-1], L[i-1,j]) & \text{if } i>0, j>0 \text{ and } X[i] \neq Y[j] \end{cases}$$

- one can formulate a DP algorithm for computing the length:
 - Time Complexity: $\Theta(m*n)$
 - Space Complexity: $\Theta(m*n)$
 - Can it be pruned?
- When is it possible to prune the space?
 - Exercise: *Modify the algorithm to compute the LCS (instead of computing the length of the LCS).*