**Wijdan Eslim - 212348718**
**Rand Salameh - 206525453**

# 2360379 - Coding and Algorithms for Memories

## SCS algorithm - final project

This project addresses the challenge of computing the Shortest Common Super-sequence (SCS) for k different sequences, while each sequence is generated by removing exactly t characters from a master sequence of length n.

**Our solution works like this:**

1. Read the number of sequences and then each sequence from input, along with the threshold t that represents the maximum number of deletions.

2. Generate all possible orderings (permutations) of the input sequences since the final result depends on the order in which the sequences are merged.

3. For each permutation, set the first sequence as the current merged result. Then, for each subsequent sequence in the permutation, merge it with the current result as follows:

   a. Compute the Longest Common Subsequence (LCS) between the current result and the next sequence using a dynamic programming approach that only fills a narrow band (of width roughly 2t+2) around the diagonal. This banded DP leverages the fact that the sequences are very similar.

   b. Reconstruct one possible LCS from the computed DP table by backtracking from the bottom-right of the table to the top-left.

   c. Build the Shortest Common Super-sequence (SCS) by interleaving the characters of the two sequences around the LCS so that all characters from both sequences are included in order.

   d. Update the current merged result to be this newly built SCS.

4. After merging all sequences for a given permutation, compare the resulting SCS with the best (shortest, and lexicographically smallest if lengths are equal) SCS found so far, and update if necessary.

5. Finally, output the best SCS and its length after examining all permutations.

**Wijdan Eslim - 212348718**
**Rand Salameh - 206525453**

# Functions explanation and Time Complexity:

`int findLCS(const string &s1, const string &s2, int t, vector<vector<int>> &dp)`

Computes the Longest Common Subsequence (LCS) using a banded dynamic programming approach. This is optimized for cases where the two strings are "close" to each other within a bounded edit distance.

Parameters:
- s1: First input string.
- s2: Second input string.
- t: The threshold defining the number of deletions
- dp: A reference to the DP table to store intermediate LCS values.

Returns: The length of the LCS.

Time Complexity: O(m × t), where m is the max length of s1 and s2, and t is the threshold for the band width (band = 2 * t + 2). The outer loop runs for m rows, and the inner loop processes approximately band columns per row, making the total work proportional to m × band.

`string backtrackLCS(const string &s1, const string &s2, const vector<vector<int>> &dp_vec)`

Reconstructs one possible LCS sequence from the computed DP table.

Parameters:
- s1: First input string.
- s2: Second input string.
- dp_vec: The computed DP table containing LCS lengths.

Returns: A string representing one LCS of s1 and s2.

Time Complexity: O(m1 + m2), The function traces back through the DP table, moving either up or left in each step, which takes at most m1 + m2 steps where m1 and m2 are the lengths of s1 and s2.

`string buildSCS(const string &s1, const string &s2, const string &lcs)`

Constructs the Shortest Common Super-sequence (SCS) from s1, s2, and their LCS.

Parameters:
- s1: First input string.
- s2: Second input string.
- lcs: The longest common subsequence of s1 and s2.

Returns: The shortest common super-sequence of s1 and s2.

Time Complexity: O(m1 + m2) where m1 and m2 are the lengths of s1 and s2. The function iterates through all characters of s1, s2, and lcs exactly once.

`string mergeTwoStrings(const string &s1, const string &s2, int t)`

Merges two strings into their shortest common super-sequence (SCS) using the three functions above.

**Wijdan Eslim - 212348718**
**Rand Salameh - 206525453**

Parameters:
- s1: First input string.
- s2: Second input string.
- t: Threshold for bandwidth in LCS calculation.

Returns: The shortest common super-sequence of s1 and s2.

Time Complexity: O(m × t), where m is the max length of s1 and s2, and t is the threshold for the band width (band = 2 * t + 2).

Explanation: O(m × t) + O(m1 + m2) + O(m1 + m2) = 3 * O(m × t) =
O(n * t) where n is the length of the master sequence.

## string mergeAllSeq(const vector<string> &seqs, int t)

Merges a sequence of strings into a single shortest common super-sequence(SCS),processing them in a chained manner.

Parameters:
- seqs: A vector of strings to merge.
- t: Threshold for band width in LCS calculation.

Returns: The shortest common super-sequence of all input strings.

Time Complexity: Depends on the number of strings (k), while each string length is at most n, and the threshold t:

1. Merging Two Strings(mergeTwoStrings): O(n × t)
2. Iterative Merging: The function merges k strings iteratively:
   - First merge: seqs[0] and seqs[1] → O(n × t).
   - Second merge: Result of the first merge with seqs[2] → O(n × t). etc.
   - Total merges: k - 1.

Total Time Complexity:
O((k - 1) × n × t) ≈ O(k × n × t).

## int main()

This main function computes and prints the Shortest Common Super-sequence(SCS) for a set of input strings and its length by trying all possible permutations of the strings and selecting the shortest SCS among them. Based on the algorithm in the first page.

Time Complexity: O(k! × k × n × t).
Explanation:
The number of permutations is k!, where k is the number of strings.
For each permutation, mergeAllSeq is called, which has a complexity of O(k × n × t). so the total time complexity is O(k! × k × n × t).