1. [4+4+4 points] Each of the following pairs of concepts are somehow connected. Describe the main connecting factors or commonalities as well as the main separating factors or differences. A few lines for each part is sufficient.

(a) Shift–And algorithm and Myers’ bitparallel algorithm.

**Solution.** Both are bitparallel algorithms for string matching, Shift-Or for exact string matching and Myers’ algorithm for approximate string matching. Both require an integer alphabet and have the same search time complexity \( O(n\lceil m/w \rceil) \) on in the worst case as well as the same preprocessing time complexity \( O(m + \sigma \lceil m/w \rceil) \). Myers’ algorithm has a better average case time complexity \( O(n\lceil k/w \rceil) \).

(b) String quicksort and string mergesort.

**Solution.** Both are algorithms for sorting strings. Both are adaptations of standard (non-string) sorting algorithms. Both work on an ordered alphabet and have time complexity \( O(n \log n + L) \) for sorting \( n \) strings with total lcp length of \( L \). Both are divide-and-conquer algorithms: string quicksort partitions by value and concatenates sorted parts, while string mergesort partitions arbitrarily and merges sorted parts. Both use lcp information. String quicksort compares strings using only one symbol at a given stage, while string mergesort compares strings using the lcp-comparison technique. String quicksort needs only \( O(\log n) \) extra space, while string mergesort needs \( O(n) \) extra space. Mergesort computes the LCP array too.

(c) Prefix doubling and induced sorting.

**Solution.** Both are suffix array construction algorithms/techniques. Prefix doubling has time complexity \( O(n \log n) \) while induced sorting has the optimal \( O(n) \) time complexity. Both assume integer alphabet \([0..n]\). Prefix doubling performs \( \log n \) iterations while induced sorting operates recursively. Both are based on lexicographical (order preserving) naming of factors.
2. **[4+4+4 points]** Give for the string set \{australia, austria, latvia, libanon, libya, lithuania, mexico, singapore, spain, sudan, sweden\}

(a) the compact trie

(b) the balanced ternary trie

(c) the LLCP and RLCP arrays for efficient binary searching in the sorted array

**Solution.**

(a)

(b)

(c) left | LLCP | mid | RLCP | right
--- | --- | --- | --- | ---
— | 0 | australia | 0 | latvia
australia | 5 | austria | 0 | latvia
— | 0 | latvia | 1 | lithuania
latvia | 1 | libanon | 2 | lithuania
libanon | 3 | libya | 2 | lithuania
— | 0 | lithuania | 0 | —
lithuania | 0 | mexico | 0 | spain
mexico | 0 | singapore | 1 | spain
lithuania | 0 | spain | 0 | —
spain | 1 | sudan | 0 | —
sudan | 1 | sweden | 0 | —
3. [6+6 points] Consider a variant of the edit distance that allows an unlimited number of insertions at the end of the string without a cost. Formally, the variant edit distance is

\[
ed'(A, B) = \min \{ ed(A, C) \mid C \text{ is a prefix of } B \},
\]

where \(ed(\cdot, \cdot)\) is the standard edit distance.

(a) Describe an algorithm that, given strings \(A\) and \(B\), computes \(ed'(A, B)\).

(b) Describe an algorithm that, given strings \(A\) and \(B\) and an integer \(k\), finds out whether \(B\) has a suffix \(B'\) such that \(ed'(A, B') \leq k\).

The time complexity should be \(O(|A||B|)\) in both cases. You may assume that any algorithms described on the lectures are known but any modifications to them should be described precisely.

Solution.

(a) Compute the same dynamic programming matrix \((d_{ij})\) as when computing the standard edit distance. Instead of returning the value at the corner \(d_{mn}\), return the minimum value at the bottom row \(\min \{d_{mj} : j \in [0..n]\}\). Each of those values corresponds to \(ed(A, C)\) for some prefix \(C\) of \(B\).

(b) Compute the same dynamic programming matrix \((g_{ij})\) as when doing approximate string matching with \(A\) as the pattern and \(B\) as the text. Return “yes” if the bottom row contains a value smaller or equal to \(k\). Each such value corresponds to a factor \(C\) of \(B\) such that \(ed(A, C) \leq k\), and such \(C\) is a prefix of some suffix \(B'\) of \(B\) such that \(ed'(A, B') \leq k\).

4. [12 points] Let \(T\) be a string of length \(n\) over an alphabet \(\Sigma\) of constant size. Describe an algorithm that finds the shortest string over the alphabet \(\Sigma\) that does not occur in \(T\). The time complexity should be \(O(n)\).

Solution. The basic idea is to perform a breadth-first traversal of the suffix tree of \(T\) and find the first locus that does not have children for all symbols. The locus may not have an explicit node. More precisely, the traversal ends when encountering an explicit node \(v\) satisfying one of the following conditions:

- \(\text{child}(v, c) = \bot\) for some symbol \(c\). In this case, the answer is \(S_v c\), where \(S_v\) is the string represented by \(v\).
- \(\text{depth}(v) > \text{depth}(u) + 1\), where \(u\) is the parent of \(v\). In this case, there is an implicit node on the edge \((u, v)\). The answer is \(S_u c d\), where \(c\) is the first symbol in the label of the edge \((u, v)\) (i.e., \(\text{child}(u, c) = v\)), and \(d\) is any symbol different from the second character of the edge label.

5S. [12 points] (Separate exam.) Describe any exact string matching algorithm covered in the study groups except Shift-And. Try to answer the following questions:

- What are the main ideas of the algorithm?
- How is the algorithm related to the algorithms described on the lectures?
- What kind of inputs the algorithm is particularly good on and why?

Solution. See study group material.
5R. [12 points] (Renewal exam only.) Any exact string matching algorithm can be used for multiple exact strings matching by searching each pattern separately, but some algorithms can be generalized to multiple patterns more efficiently. For example, the Aho–Corasick algorithm is a generalization of the Morris–Pratt algorithm. Describe such a generalization for some exact string matching algorithm other than (Knuth–)Morris–Pratt. You can choose any of the algorithms in the lectures but the asymptotic time complexity of your solution should be better than searching each pattern separately using the same algorithm.

Solution. See Exercise 4.5.