Algorithms and Data Structures II (1DL231)
Uppsala University — Autumn 2022

Assignment 2

Based on assignments by Pierre Flener,
but revised by Frej Knutar Lewander and Justin Pearson

— Deadline: 15:00 on Friday 1st December 2023 —

It is strongly recommended to read the *Submission Instructions* and *Grading Rules* at the end of this document even before attempting to solve the following problems. It is also strongly recommended to prepare and attend the help sessions.

For this assignment you will need the two Python skeleton files `difference.py` and `recompute_mst.py`

Problem 1: Search-String Replacement

A useful feature for a search engine is to suggest a replacement string when a search string given by the user is not known to the search engine. In order to suggest and rank replacement strings, the search engine must have some measure of the minimum difference between the given search string and a possible replacement string. For example, over the alphabet $\mathcal{A} = \{\mathrm{A}, \ldots, \mathrm{Z}\}$, let the user's search string be $s = \mathrm{DINAMCK}$ and let a suggested replacement string be $r = \mathrm{DYNAMIC}$. The *minimum difference* of strings s and r is the minimum cost of changes transforming s into r, where a *change* is either altering a character in s in order to get the corresponding character in r, or skipping a character in s or r. A *positioning* of two strings is a way of matching them up by writing them in columns, using a dash (–) to indicate that a character is skipped. For example:

D	I	N	A	M	–	C	K
D	Y	N	A	M	I	C	–

The *difference* of a positioning is then the sum of the resemblance costs of the character pairs in each column of the positioning, as given by a resemblance matrix $\mathcal{R}$. For an alphabet $\mathcal{A}$, we have that $\mathcal{R}$ is an $(|\mathcal{A}|+1) \times (|\mathcal{A}|+1)$ matrix, as it must include the dash in addition to the $|\mathcal{A}|$ characters of the alphabet. For example, the positioning above has a difference of:

$$\mathcal{R}[\mathrm{D,D}] + \mathcal{R}[\mathrm{I,Y}] + \mathcal{R}[\mathrm{N,N}] + \mathcal{R}[\mathrm{A,A}] + \mathcal{R}[\mathrm{M,M}] + \mathcal{R}[\text{–,I}] + \mathcal{R}[\mathrm{C,C}] + \mathcal{R}[\mathrm{K,}\text{–}]$$

For example, if $\mathcal{R}[x, y] = 1$ for all $x, y \in \mathcal{A} \cup \{\text{–}\}$ with $x \neq y$ and if $\mathcal{R}[x, x] = 0$ for all $x \in \mathcal{A} \cup \{\text{–}\}$, then the difference of a positioning is the *number* of changes; another resemblance matrix could store the Manhattan distances on a QWERTY keyboard between characters of the alphabet (see the skeleton code).

Given two strings s and r of possibly different lengths over an alphabet $\mathcal{A}$ that does not contain the dash character, and given an $(|\mathcal{A}|+1) \times (|\mathcal{A}|+1)$ resemblance matrix $\mathcal{R}$ of integers, which ***cannot*** be assumed to be symmetric, perform the following tasks:

A. Give a recursive equation for a parameterised quantity after stating its meaning in terms of all its parameters (do not rename the problem parameters s, r, $\mathcal{A}$, and $\mathcal{R}$). Complete the following sub-tasks:

   (a) Give the base case(s) of the recursive equation.

   (b) Give the recursive case(s) of the recursive equation.

   (c) Use the equation to justify that the problem of computing the minimum difference of s and r relative $\mathcal{R}$ has optimal substructure and overlapping subproblems, so that dynamic programming is applicable to it.

B. Implement an *efficient* dynamic programming algorithm for this problem as a Python function *min_difference*$(s, r, \mathcal{R})$, assuming that the *last* row and *last* column of $\mathcal{R}$ pertain to the dash character.

   Write a short description of which dynamic approach you chose (bottom-up recursive or top-down iterative) you used and how your code implements it. Your implemented function must pass ***all*** corresponding unit tests in the skeleton code file.

C. Extend your algorithm from Task B to return also a positioning for the minimum difference. Implement the extended algorithm as a Python function *min_difference_align*$(s, r, \mathcal{R})$. Your implemented function must pass ***all*** corresponding unit tests in the skeleton code file.

D. Argue that the time complexity of your extended algorithm is $\mathcal{O}(|s| \cdot |r|)$.

If you pass only Tasks A to B (and their sub-tasks), then your score is up to 3 points for the Problem. If you pass only Tasks A to C (and their sub-tasks), then your score is up to 4 points for the Problem. If you pass Tasks A to D (and their sub-tasks), then your score is up to 5 points for the Problem.

We are ***not*** implying that search engines actually use such a dynamic programming algorithm for suggesting search-string replacements.

Problem 2: Recomputing a Minimum Spanning Tree

Given a connected, weighted, undirected graph $G = (V, E)$ with ***non-negative*** edge weights, as well as a minimum(-weight) spanning tree $T = (V, E')$ of G, with $E' \subseteq E$, consider the problem of incrementally updating T if the weight of a particular edge $e \in E$ is updated from $w(e)$ to $\widehat{w}(e)$. There are four cases:

1. $e \notin E'$ and $\widehat{w}(e) > w(e)$
2. $e \notin E'$ and $\widehat{w}(e) < w(e)$
3. $e \in E'$ and $\widehat{w}(e) < w(e)$
4. $e \in E'$ and $\widehat{w}(e) > w(e)$

Perform the following tasks:

A. For each of the four cases: describe in plain English with mathematical notation an efficient algorithm for updating the minimum spanning tree, and argue that the algorithm is correct and has a time complexity of $\mathcal{O}(1)$ or $\mathcal{O}(|V|)$ or $\mathcal{O}(|E|)$.

B. Choose two of the four cases. for ***each*** chosen case, say case $i \in 1..4$, implement your algorithm as a Python function $update\_MST\_i(G, T, e, w)$ for $w = \widehat{w}(e)$. For each of your implemented functions, if the edge e_r was removed from T and the edge e_a was added to T, then the function should return the tuple (e_r, e_a), otherwise it should return the tuple $(None, None)$.

C. For another case that was not chosen in Task B, say case $j \in 1..4$, implement your algorithm as a Python function $update\_MST\_j(G, T, e, w)$ for $w = \widehat{w}(e)$. If the edge e_r was removed from T and the edge e_a was added to T, then your implemented function should return the tuple (e_r, e_a), otherwise it should return the tuple $(None, None)$.

D. Briefly describe a real-world situation where (a variation) of Recomputing a Minimum Spanning Tree can occur.

If you pass only Tasks A to B, then your score is up to 3 points for the Problem. If you pass only Tasks A to C, then your score is up to 4 points for the Problem. If you pass Tasks A to D, then your score is up to 5 points for the Problem.

Submission Instructions

- Identify yourself inside the report and ***all*** code.
- State the problem number and task identifier for each answer in the report.

- Take Part 1 of the demo report at `http://user.it.uu.se/~justin/Hugo/courses/ad2/demorep` as a *strict* guideline for document structure and as an indication of its expected quality of content.
- Comment *each* function according to the AD2 coding convention at `http://user.it.uu.se/~justin/Hugo/courses/ad2/codeconv`.
- Test *each* function against *all* the provided unit tests.
- Write *clear* task answers, source code, and comments: write with the precision that you would expect from a textbook.
- Justify *all* task answers, except where explicitly not required.
- State in the report *all* assumptions you make that are not in this document. Every legally re-used help function of Python can be assumed to have the complexity given in the textbook, even if an analysis of its source code would reveal that it has a worse complexity.
- *Thoroughly* proofread, spellcheck, and grammar-check the report.
- Match *exactly* the uppercase, lowercase, and layout conventions of any filenames and I/O texts imposed by the tasks, as we will process submitted source code automatically.
- Do *not* rename any of the provided skeleton codes, for the same reason.
- Import the commented Python source-code files *also* into the report: for brevity, it is allowed to import only the lines between the copyright notice and the unit tests.
- Produce the report as a *single* file in PDF format; all other formats will be rejected.
- Remember that when submitting you implicitly certify (a) that your report and all its uploaded attachments were produced solely by you, except where explicitly stated otherwise and clearly referenced, (b) that you can explain any part starting from the moment of submitting your report, and (c) that your report and attachments are not freely accessible on a public repository.
- Submit the solution files (one report and up to two Python source-code files) without folder structure and without compression via *Studium*

Grading Rules

For each problem: if (for all of the following statements)

- there are no runtime errors when running the code under Python 3 on a Linux computer of the IT department, and
- the code passes *all* of the provided unit tests and *all* of our grading tests,

then your report will be graded for the problem (continue reading). Otherwise, the final score is automatically 0 points for the problem, and the report will not be graded for the problem. Furthermore, if (for all of the following statements)

- the requested source code exists in a file with *exactly* the name of the corresponding skeleton code (except for any characters introduced by studium);

- the code imports *only* the libraries imported by the skeleton code;
- the code has the comments prescribed by the AD2 coding convention for *all* the functions; and
- the source code features a *serious* attempt at algorithm analysis,

Then no points will be deducted from the final score for the problem. Otherwise, points will be deducted from the final score of the problem, at the discretion of the assistant.

Considering there are three help sessions for each assignment, you must earn at least 3 points (of 10) on each assignment until the end of its grading session, including at least 1 point (of 5) on each problem and at least 15 points (of 30) over all three assignments, in order to pass the *Assignments* part (2 credits) of the course.