Algorithms and Data Structures II (1DL231)
Uppsala University – Autumn 2019
Assignment 2

Prepared by Pierre Flener

— Deadline: 13:00 on Friday 6 December 2019 —

It is strongly recommended to read the *Submission Instructions* and *Grading Rules* at the end of this assignment statement ***before*** attempting to solve the following problems. It is also strongly recommended to prepare and attend the help sessions, as huge time savings may ensue.

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 u = DINAMCK and let a suggested replacement string be r = DYNAMIC. The *minimum difference* of strings u and r is the minimum cost of changes transforming u into r, where a *change* is either altering a character in u in order to get the corresponding character in r, or skipping a character in u 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 u 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 the minimum difference of u and r, including the semantics of all the parameters. Use it to justify that dynamic programming is applicable to the problem of computing this minimum difference.

B. Design (including the choice between top-down recursive and bottom-up iterative) and implement an efficient dynamic programming algorithm for this problem as a Python function *min_difference*$(u, r, \mathcal{R})$, assuming that the *last* row and *last* column of $\mathcal{R}$ pertain to the dash character.

C. Extend your function from task B to return also a positioning for the minimum difference. Implement the extended algorithm as a Python function *min_difference_align*$(u, r, \mathcal{R})$.

D. Argue that your (extended) algorithm has a time complexity of $\mathcal{O}(|u| \cdot |r|)$.

Solo teams may omit task C. (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 each algorithm is correct and has a time complexity of $\mathcal{O}(|V|)$ or $\mathcal{O}(|E|)$.

B. For ***at least one*** case that does ***not*** take constant time, say case $i \in 1..4$, implement your algorithm as a Python function *update_MST_i*(G, T, e, w) for $w = \widehat{w}(e)$. If you give functions for multiple values of i, then indicate which one you want to be graded, else we choose the one with the lowest i. Note that our skeleton code and grading are based on NetworkX version 1.11, ***not*** the current version 2.x.

Solo teams may omit task B.

Submission Instructions

- Identify the team members and state the team number inside the report.
- 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/~pierref/courses/AD2/demoReport` 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/~pierref/courses/AD2/codeConv.html`.

- Write clear task answers, source code, and comments.
- Justify all task answers, except where explicitly not required.
- State in the report any assumptions you make that are not in this assignment statement. Any 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 source code automatically.
- Import the commented Python source-code files ***also*** into the report.
- 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 your team, except where explicitly stated otherwise and clearly referenced, (b) that each teammate can individually 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 (by only ***one*** of the teammates) the solution files (one report and up to two Python source-code files) ***without*** folder structure and ***without*** compression via the *Student Portal*, whose clock may differ from yours, by the given ***hard*** deadline.

Grading Rules

For each problem: ***If*** the requested source code exists in a file with exactly the name of the corresponding skeleton code, ***and*** it depends only on the libraries imported by the skeleton code, ***and*** it runs without runtime errors under version 2.7.15 of Python (use `python`; *not* the current version 3.x.y) and version 1.11 of NetworkX (*not* the current version 2.x), ***and*** it produces correct outputs for some of our grading tests in reasonable time on the Linux computers of the IT department, ***and*** it has the comments prescribed by the AD2 coding convention, ***and*** it features a serious attempt at algorithm analysis, ***then*** you get at least 1 point (of 5), ***otherwise*** you get 0 points. Furthermore:

- If your function has a ***reasonable*** algorithm and ***passes most*** of our grading tests, ***and*** your report is ***complete***, then you get a final score of 3 or 4 or 5 points, depending also on the quality of the Python source code comments and the report part for this problem; you are not invited to the grading session for this problem.
- If your function has an ***unreasonable*** algorithm *or* ***fails many*** of our grading tests, ***or*** your report is ***incomplete***, then you get an initial score of 1 or 2 points, depending also on the quality of the Python source code comments and the report part for this problem; you are invited to the grading session for this problem, where you can try and increase your initial score by 1 point into your final score.

However, ***if*** the coding convention is insufficiently followed ***or*** the assistants figure out a minor fix that is needed to make your source code run as per our instructions, ***then***, instead of giving

0 points up front, the assistants may at their discretion deduct 1 point from the score earned otherwise.

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