Algorithms and Data Structures II (1DL231)
Uppsala University – Autumn 2017
Assignment 2

Prepared by P. Flener, A. Ek, C. Pérez Penichet, H.-P. Vo, and K. Winblad

— Deadline: 13:00 on Friday 2017-12-08 —

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

Problem 1: Search String Replacement (*mandatory!*)

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 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 *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 a $(|\mathcal{A}| + 1) \times (|\mathcal{A}| + 1)$ resemblance matrix $\mathcal{R}$ of integers, which ***cannot*** be assumed to be symmetric, perform the following tasks:

a. Give a recursive equation for the difference of u and r. Use it to justify that dynamic programming is applicable to the problem of computing this difference.

b. Design 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 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: Ring Detection in Graphs

In an undirected graph, a path $\langle v_0, v_1, \ldots, v_k \rangle$ forms a *ring* if the vertices v_0 and v_k are equal ***and all edges on the path are distinct***:

$$\forall i, j \in 0\,..\,k-1 : i \neq j \Rightarrow (v_i, v_{i+1}) \neq (v_j, v_{j+1})$$

Perform the following tasks:

a. Design and implement an efficient algorithm as a Python function *ring*(G) that returns *True* if and only if there exists a ring in the undirected graph $G = (V, E)$. Two points will be deducted from your score if your algorithm deletes vertices or edges; in that case, deletions must be made on a ***copy*** of G in order to comply with the style of graph algorithms in CLRS3. You can ***not*** assume that G is connected. Also recall (see Appendix B.4 of CLRS3) that in an undirected graph the edges (u, v) and (v, u) are considered to be the same edge and that self-loops (edges from a vertex to itself) are forbidden.

b. Extend your function from task a in order to return also a ring, if one exists. Implement your extended algorithm as the Python function *ring_extended*(G).

c. Argue that your (extended) algorithm has a time complexity of $\mathcal{O}(|V|)$, independent of $|E|$.

Solo teams may omit task b.

Problem 3: Recomputing a Minimum Spanning Tree

Given a connected, weighted, undirected graph $G = (V, E)$ with ***non-negative*** number 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:

a. $e \notin E'$ and $\widehat{w}(e) > w(e)$

b. $e \notin E'$ and $\widehat{w}(e) < w(e)$

c. $e \in E'$ and $\widehat{w}(e) < w(e)$

d. $e \in E'$ 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 $w = \widehat{w}(e)$.

Solo teams may omit task b.

Submission Instructions

All task answers, including documented source code, ***must*** be in a ***single*** report in ***PDF*** format; all other formats are rejected. Furthermore:

- Identify the team members and state the team number inside the report.
- Certify that your report and all its uploaded attachments were produced solely by your team, except where explicitly stated otherwise and clearly referenced, that each teammate can individually explain any part starting from the moment of submitting your report, and that your report and attachments are not and will not be freely accessible on a public repository. Without such an honour declaration, your report will ***not*** be graded.
- State the problem number and task identifier for each answer in the report.
- Take Section 1 of the demo report at `http://user.it.uu.se/~pierref/courses/AD2/demoReport` as a ***strict*** guideline for document structure and content, as well as an indication of its expected quality.
- Comment each function according to the AD2 coding convention at the *Student Portal*.
- Write clear task answers, source code, and comments.
- Justify all task answers, except where explicitly not required.
- State any assumptions you make that are not in this document. 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 your report.
- Match exactly the uppercase, lowercase, and layout conventions of any filenames and I/O texts imposed by the tasks, as we will process your source code automatically.
- Make a sanity check of all your Python functions on *random* tests via the server at `http://ad2checker.it.uu.se`. Beware: passing random tests does ***not*** guarantee that your code will pass the *boundary* and *stress* tests of our grading test suite (see below), hence you are ***strongly*** encouraged to devise your own *boundary* and *stress* tests.

Only ***one*** of the teammates submits the solution files (one PDF report with answers to ***all*** the tasks, plus up to three commented Python source code files, which are ***also*** imported into the report), ***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.∗ of Python (use `python`), ***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 at the *Student Portal*, ***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 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 mark 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 mark by 1 point into your final mark.

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 above, ***then***, instead of giving 0 points up front, the assistants may at their discretion deduct 1 point from the mark earned otherwise.

Considering that there are three help sessions for each assignment, you must get minimum 5 points (of 15) on each assignment until the end of its grading session, including minimum 1 point (of 5) on each mandatory problem, and minimum 23 points (of 45) over all three assignments; otherwise you fail the *Assignments* part of the course.