STAT 534 Homework 5
Due May 20, 2019
©Marina Meilă
mmp@stat.washington.edu

Problem 1 – Data structures for disjoint sets – Part II

You will write a program in the file that applies the DSF functions to the `statisticiansA-M.txt`.

Submission details Write a Python program called `hw5pb1.py` that imports the code from Homework 4 Problem 1 (function definitions) as a module. Submit both your module and `hw5pb1.py`, but do **not** submit as a compressed format (.zip, .rar, etc.). The filename of your module should be `<last name><first name>_disjointsets.py`, where your first and last name are **as they appear on Canvas** and the file name is **all lowercase**. For example, if your name is "John Smith," but your name on Canvas is "Jonathan Smith," then the filename for your module should be `smithjonathan_disjointsets.py`
Hence, the first line of your `hw5pb1.py` may look something like:

```
from smithjonathan_disjointsets import *
```

or

```
import smithjonathan_disjointsets as ds
```

Failing to follow the above instructions may result in losing some or all of the points for this problem.

a.In `hw5pb1.py`, implement the following:

1. Read in to Python the (non-truncated) last names from the file "statisticiansA-M.txt".
2. Assign each name a number from 0 to $n-1$, where $n = 393$ is the total number of unique statisticians in the file. We refer to this as the *ID* i of the statistician.
3. Using the ID's of each statistician as the node labels, initialize a DSF using `make_set` operations, with n nodes corresponding to the n statisticians.
4. Write a function `find_set_statistician` that:
   (i) Takes the last name of a statistician as input

(ii) Finds the node s corresponding to the input name. If there is no node corresponding to the input name, returns `null`

(iii) Finds the representative r of node s, if s not `null`.

(iv) Returns the last name of the statistician corresponding to r. If s is `null` returns `'None'`. Return the last name of repr

5. Read in the list of edges from file `hw5-statisticians-edges.dat`, which contains a pair of ID's in plain text, on each line.

6. Use these edges to perform the following operations on the DSF initialized in step 3

```
union( x1, y1 )
union( x2, y2 )
...
union( xm, ym )
```

7. At the end of your program, write the following test cases *exactly as they appear*:

```
print(find_set_statistician('Blackwell'))
print(find_set_statistician('Bottou'))
print(find_set_statistician('Brad'))
print(find_set_statistician('Breslow'))
print(find_set_statistician('Wellner'))
print(find_set_statistician('Laird'))
print(find_set_statistician('Fisher'))
print(find_set_statistician('Holmes'))
```

b. Describe how you implemented the Disjoint Set Forest (DSF) data structure in **a** the `class Node`. E.g. with an array, a dictionary, and what do the entries represent (1-2 paragraphs)?

Describe in enough detail that we can evaluate if the functions MAKE-SET, FIND-SET, UNION operating on your data structure achieve the asymptotic running time of their pseudocode versions, assuming that the code will be required to perform a sequence of FIND-SET and UNION calls with this DSF. *Small constant differences can be ignored.*

Problem 2 – Edit distance between two strings Problem 15–3 CRLS page 365 *modified as follows:*

The questions in CRLS 15–3 are reformulated here to make them more precise and to give a few hints about their solutions. Consider the original questions **a, b** *in CRLS replaced by the questions below.*

Given two sequences $x[1 \ldots m]$, $y[1 \ldots n]$ and a set of transformation-operation costs, the *edit distantce* from x to y is the cost of the least expensive operation sequence that transforms x to y.

Use the following costs in your answers to **b, c, d**.

```
copy     0
replace  1
delete   2
insert   2
twiddle  1
kill     3
```

a. First, describe the optimality structure of the problem: if we know an optimal solution to the string edit distance (SED) problem, then are there any parts of the solution that are optimal solutions to SED subproblems? Give an analog of Theorem 15.1 and explain why it should hold.

b. SED without twiddle: Give a dynamic programming algorithm (in pseudocode) that solves the SED problem and prints an optimal operation sequence, in the case when the twiddle operation is not an option. Be sure to also explain your idea in words. What are the running time and the space requirements of your algorithm (in $\mathcal{O}$ notation) as a function of m, n? [**Hint:** This problem is very similar to the LCS problem. You will have to set up and fill a pair of tables similar to tables c, b in the LCS algorithm. In the following, I will refer to the tables associated to the SED algorithm as c and b.]

c. SED with twiddle: Now modify the algorithm you obtained in **b** to include the option of performing a twiddle. What are the running time and space requirements of the modified algorithm?

d. Demonstrate the algorithm in **c.** in the case SILK $\rightarrow$ SICKLE:

- construct and fill the c, b tables
- list the sequence of operations and the total cost

e. Extra credit Explain how to cast the problem of finding an optimal alignment described in CRLS pp 366–367 as an edit distance problem using a subset of the transformation operations copy, replace, delete, insert, twiddle and kill. Assign each operation the cost that you'll find necessary.