5LN715 Written Exam 2020-03-27

Instruction:

Submit the solutions as one pdf-document in the Student Portal (as an assignment). Deadline 2020-03-27 20.00. **By submitting the solution you certify that it is the product of your own creativity and skills. You are not allowed to take help or work with anyone else.** However, you can use available online and printed material.

Only short answers are necessary, but should be exact, and neatly and clearly formatted, as details matter. Use of Python 3 is presupposed. Remember that indentation is significant! Of course, different solutions are often possible. Try to find the best ones!

The tasks are intended to be clear, well-defined, and straightforward. However, if some task seems ambiguous or open to you, solve it in the way you find most reasonable (most relevant given the course content), and write a note about how you interpreted the instruction. We will also monitor the Student Portal forum during the morning. You are welcome to ask for clarifications, but please don't reveal the solutions you have in mind.

The grade thresholds are as follows: G: 50% and VG: 75%. (The examination of the course also involves a series of assignments, as you know.)

GOOD LUCK!

Question 1 [max 6 p]

Here is some code handling blogs, blog posts and comments on blogs. A blog has an owner who writes all the posts. The posts have a title and some text. There can be comments on blog posts, and also comments on other comments. A comment has an author and a text.

```
class Blog:
    def __init__(self, name, owner):
        self.name = name
        self.owner = owner
        self.posts = []

    def add_post(self, post):
        self.posts.append(post)

    def output_with_comments(self):
        title = f'{self.name} (by {self.owner})'
        print(title)
        print('=' * len(title))
        print()
        for p in self.posts:
            p.output_with_comments()
        print()
```

```
class BlogPost:
    def __init__(self, title, text=''):
        self.title = title
        self.text = text
        self.comments = []

    def add_comment(self, comment):
        self.comments.append(comment)

    def output_with_comments(self):
        print(self.title)
        print('-' * len(self.title))
        print(self.text)
        print()
        for c in self.comments:
            c.output_with_comments(level=1)
        print()


class Comment:
    indent_marker = '> '

    def __init__(self, author, text=''):
        self.author = author
        self.text = text
        self.comments = []

    def add_comment(self, comment):
        self.comments.append(comment)

    def output_with_comments(self, level=0):
        indent = self.indent_marker * level
        print(f'{indent}[Comment by {self.author}:]')
        print(f'{indent}{self.text}')
        for c in self.comments:
            c.output_with_comments(level=level+1)
```

The `output_with_comments` method on a blog might print something like the text in the following box. (If this was real it would probably output HTML suitable for the web instead, but for now it is just plain text.)

```
My Thoughts (by Huey)
=====================

New blog
--------
I've started a blog. This is the first post.

> [Comment by Dewey:]
> I look forward to reading your blog!

Thought for today
-----------------
Flowers are nice. Do you also like them?

> [Comment by Dewey:]
> I agree.
> [Comment by Louie:]
> That is so wrong!
> > [Comment by Dewey:]
> > You are so negative, Louie!
> > > [Comment by Louie:]
> > > I just don't agree. Flowers are bad!
> [Comment by Huey:]
> I forgot to say that I especially like sunflowers.

Ending the blog
---------------
This is my last entry here. It has been fun!

> [Comment by Louie:]
> Actually I liked your blog.
```

(a) Name one *class variable* in this example? **(1 p)**

(b) In the example output, for which shown objects is it true that the length of its instance variable `comments` is 1? **(1 p)**

(c) Write a `posts_with_text` method for Blog which returns a list of blog posts in that blog where the title or the text of the post contains that (exact) text. **(2 p)**

(d) There should be a variant of `BlogPost` called `BlogPostForbiddingComments` which is like a normal blog post, except that if you try to add a comment to it it should yield an error by doing

```
raise TypeError
```

Write the full code for that, inheriting from the given class! **(2 p)**

(a) The only one is `indent_marker`.

(b) The blog entries 'New blog' and 'Last entry'; and the comments 'That is so wrong!' and 'You are so negative, Louie!'.

(c)
```
def posts_with_text(self, text):
    posts = []
    for p in self.posts:
        if text in p.title or text in p.text:
            posts.append(p)
    return posts
```

or,

```
def posts_with_text(self, text):
    return [p for p in self.posts if text in p.title or text in p.text]
```

(d)
```
class BlogPostForbiddingComments(BlogPost):
    def add_comment(self, comment):
        raise TypeError
```

Question 2 [max 4×2 p]

Determine the worst case time complexity for each code snippet below, and support your answer with a short argument. Assume that `numbers` is an array (of length n) containing integers and that `m` is a positive integer.

(a)

```
for i in numbers:
    print(i)
    for j in range(m):
        print(m)
    for k in range(50000):
        print(k*i)
for i in numbers:
    print("hello world")
```

(b)

```
i = len(numbers)
while i > 0:
    numbers[i] = numbers[i]*5
    i -= 3
```

(c)

```
for j in numbers:
    i = len(numbers)
    while i > 0:
        print(j)
        i = round(i/2.8)
```

(d)

```
for i in range(len(numbers)):
    for j in range(i):
        k = numbers[i] + numbers[j]
```

a: $O(n*m)$ – For the first part, the outer loop is over n, the two inner loops are over m and 50,000 respectively. The loop over 50,000 is ignored, since it is a constant (even though it is large). The complexity of this loop is thus n multiplied by m, i.e. $O(n*m)$. The second loop is over n, and has an complexity of $O(n)$, which is dominated by the first loop, and does not affect the overall complexity.
b: $O(n)$ – The loop actually runs $n/3$ times, since we decrease i with 3 each loop. 3 is a constant, and the complexity is thus linear.
c: $O(n\log n)$ – The outer loop is over n, and in the inner loop we keep dividing by 2.8, which gives a logarithmic complexity. The fact that we divide by 2.8 and not 2 does not change the fact that it is logarithmic. Since the loops are nested we need to multiply n and log n.
d: $O(n^2)$ – The outer loop runs over n,and the inner loop runs 0 times the first time, 1 time the next time, up to n times the last time. This is an example of triangular numbers, which means that the loop runs $1+2+\cdots+n = (n(n+1))/2$ times, giving a complexity of $O(n^2)$.

Question 3 [max 10 p]

Write code for the two methods

```
printSkip(self)
removeValue(self, value)
```

that should be part of the LinkedList class below, which represent a singly-linked list. Each node has an integer representing a *skip*, and a value which should be printed, called data. You are not allowed to change the given code.

The `printSkip` method should print the value in the first node, then skip as many nodes as indicated by the `skip` in the first node, print the next value, and skip the number of nodes indcated by that `skip`, and so on. When it reaches the end of the list, it should just stop.

`removeValue` takes a value as input, and should remove all nodes containing that value stored in `data`.

Note that each method should go through the list only one time, i.e. have time complexity $O(n)$. For full points, we expect as simple a solution as possible (without any unnecessary if-clauses, for instance).
As an example, a linked list containing the nodes (data, skip):

```
("a",2) -> ("b",5) -> ("c",1) -> ("d", 3) -> ("e",6) -> ("a",1) -> ("g",6) -> ("a", 3)
```

Should printSkip "a, c, d, g" and if we apply `removeValue` with "a", it should result in this list:

```
("b",5) -> ("c",1) -> ("d", 3) -> ("e",6) -> ("g",6)
```

```
#Node class:
class Node:
    def __init__(self, data, skip, next):
        self.next = next
        self.data = data
        self.skip = skip

#List class
class LinkedList:
    def __init__(self):
        self.head = None

    def insert(self, data, skip):
        self.head = Node(data, skip, self.head)
```

```
def printSkip(self):
    current = self.head
    step = 0
    skip = 1
    while current:
        step += 1
        if step == skip:
            print (current.data)
            step = 0
            skip = current.skip
        current = current.next

def removeValue(self, value):
    while self.head and self.head.data == value:
        self.head = self.head.next
    if self.head:
        current = self.head
        while current.next:
            if current.next.data == value:
                current.next = current.next.next
            else:
                current = current.next
```

Question 4 [max 10 p]

Below you will find the timings of two different sorting algorithms, sortX and sortY, for arrays of lengths 1,000, 10,000, and 100,000. The arrays contain integers from 0 to length-1, and have been manipulated before sorting using either of two strategies `shuffle1` or `shuffle2` outlined below. You should discuss the complexity and other features of these two sorting algorithms, given the times in these different settings. Make a guess and motivate it, about which sorting algorithms you think were used. (Note, the discussion and motivation is much more important than guessing correctly).

Timings:

```
sortX, shuffle1, 1000, time: 0.032445
sortY, shuffle1, 1000, time: 0.003253
sortX, shuffle2, 1000, time: 0.001209
sortY, shuffle2, 1000, time: 0.002829
sortX, shuffle1, 10000, time: 3.270695
sortY, shuffle1, 10000, time: 0.044512
sortX, shuffle2, 10000, time: 0.015209
sortY, shuffle2, 10000, time: 0.037252
sortX, shuffle1, 100000, time: 331.031591
sortY, shuffle1, 100000, time: 0.558342
sortX, shuffle2, 100000, time: 0.106189
sortY, shuffle2, 100000, time: 0.443340
```

Shuffling strategies:

```
#list initialization:
myList = list(range(length))

def shuffle1(l):
    for i in range(len(l)):
        r = randint(0,len(l)-1)
        (l[i],l[r]) = (l[r],l[i])


def shuffle2(l):
    for i in range(10):
        r = randint(0,len(l)-1)
        s = randint(0,len(l)-1)
        (l[s],l[r]) = (l[r],l[s])
```

sortX is insertion sort and sortY is merge sort. There is a clear difference in behaviour between the two algorithms for the cases when the array is completely scrambled (shuffle1), and where it is nearly sorted (shuffle2). sortY is clearly faster for a scrambled array, and the difference to sortX grows with the size of the array. For a nearly sorted array, though, sortX is faster than sortY. The fact that sortX is much faster for a nearly sorted than for a scrambled array must means that it is adaptive, i.e. that it takes advantage of the initial order of the array. sortY is only slightly faster for a nearly sorted than a scrambled array, and the complexity does not seem to change, indicating that it is not adaptive. The increase in time with the size for sortX seems to be linear for a nearly sorted array and polynomial for a scrambled array. This fits insertion sort which is adaptive with a worst case time complexity of $O(n^2)$ and a best case time complexity of $O(n)$ (and would also be a reasonable fit for improved bubble sort among the algorithms discussed in class). sortY grows more slowly than sortX, but a bit faster than linearly, with size, so it seems reasonable that it might have time complexity $O(n \log n)$, and it is not adaptive. This fits merge sort (and would also fit quick sort with a simple choice of pivot, among the algorithms discussed in the course).
Note: guessing improved bubble sort or quick sort should not decrease the score, given a good motivation. Guessing correctly without a good motivation should not give a full score.

Question 5 [max 10 p]

Assume that you want to write a program that will process a very large file of word-POS pairs. The output of the program should be a frequency list of each word-POS pair, with the percentage of a POS among all options for that word. For example, if the input file contains the word *book* 10 times, 8 times as a noun and 2 times as a verb, and the word *pen* 3 times, always as noun, your program should output: "book NOUN 8 (80%), book VERB 2 (20%), pen NOUN, 2 (100%)". The output should be sorted alphabetically based on the word and POS. The input format of the file to read is not really important for the task, but it makes it easy to extract word+POS pairs. The file will be very large, containing a high number of both types and tokens. You can assume that the distribution of words and POS-tags are representative of English.

Discuss at least two different options for data structures that you could use for solving this task. Try to come up with as efficient data structure options as you can. Describe how you would store the information using each data structure. Then, go through all operations needed for storing the data and printing the asked for information. For each operation, discuss what the time complexity is of that operation with each of your data structure options. Note that the exact operations might be slightly different for the two data structures. Also note that it is possible, but not necessary, that your proposed data structure solutions use more than one data structure each.

Note: you do not have to write any code here. Only describe how you would have organised the data structures used in your code, and the time complexity issues.

I will discuss the two options: dictionary+linked list and binary search tree.
I will use the following constants: n the number of word tokens in the file and m the number of word types (unique words). p the number of POS-tags, r the average number of POS-tags per word.
Using a dictionary I would have an outer dictionary with each word as a key. As the value for each key, I would have a sorted linked list containing each possible POS-tag with the frequency for the word+POS pair. The reason for an outer dictionary is that the number of word types is high, and access to storage is fast in a dictionary. On the other hand, the number of POS-tags per word is low, and a linked list can be kept in sorted order with a low cost in such a case. The operations needed when reading words is to increase the frequency word+POS combinations n times. The cost of each such operation is $O(1)$ for finding or inserting the word into the dictionary. For finding the correct POS-tag, the cost is in the worst case $O(p)$, but on average it is $O(r)$, and each word type typically has few possible POS-tags. The POS-tags can easily be inserted sorted in the linked list, withut any additional cost. The cost for each new word+POS pair read, is thus $O(r)$ on average. And for reading all words, it is $O(n*r)$. To be able to print the words alphabetically sorted, there is a need for sorting the dictionary keys. The cost for this is $O(n\log n)$ using an efficient sorting algorithm like merge sort. Printing and calculating the percentages can be done in one pass through the dictionary. For each word, calculating the percentage for each tags takes $O(p)$ worst case and $O(r)$ average case. and printing takes the same. So the full cost of printing and calcualating for all words is $O(n*r)$. In summary The cost for inserting, calculating and printing is $O(n*r)$ (based on average number of POS-tags), and for sorting $O(n\log n)$.
My other option is to use binary search trees. Here I choose to use nested trees for storing the words, and for storing POS for each word. I assume I use some kind of balanced tree like AVL-trees, which gives access in $O(\log n)$. In this case there is no need for sorting, since the AVL-tree is already sorted. The cost for finding and inserting each word is $O(\log m)$ and the cost for finding the POS for each word is $O(\log p)$ worst case and $O(\log r)$ average case. The cost for inserting a single word is thus $O(\log m+\log r)$ average case, which gives a total cost of word insertion of $O(n*(\log m+\log r))$ The cost for printing and calculating the percentages for each word is $O(\log r)$ respectively, since we have to go through all POS-tags. And doing this for all words is $O(n\log r)$.
In summary, access is faster in a dictionary than a binary search tree, but the dictionary requires sorting, which is expensive. A linked list is slower than a binary search tree, While the lookup of POS-tags is faster here, $O(r)$ instead of $O(\log r)$ for a linked list, but it makes little difference in practice, since r is very small but for a small set.
Comments: other data structures are also possible. A POS dictionary with word lists is not a very good solution. The same can be said about plain lists or lists of lists.

Question 6 [max 10×1 p]

Consider the `numpy.array` A below:

```
A = np.array([[44, 35, 72, 55, 15, 69],
              [23, 42, 82, 74, 11, 61],
              [77, 50, 20, 54,  7, 49]])
```

i. What is the dimensionality of A?

ii. What is the shape?

iii. How can one retrieve `[23, 42, 82, 74, 11, 61]` from A?

iv. ... `[72, 82, 20]`?

v. ... `[11]`?

vi. ... `[42, 82, 50, 20]`?

vii. How can we permute A into the following array?

```
array([44, 35, 72, 55, 15, 69, 23, 42, 82,
       74, 11, 61, 77, 50, 20, 54,  7, 49])
```

viii. How can we permute A into the following array?

```
array([[44, 35],
       [72, 55],
       [15, 69],
       [23, 42],
       [82, 74],
       [11, 61],
       [77, 50],
       [20, 54],
       [ 7, 49]])
```

ix. How can one obtain the mean of A and subtract it from every element, so that we're left with the following array?

```
    array([[ -2.66666667, -11.66666667, 25.33333333, 8.33333333,
             -31.66666667,  22.33333333],
           [-23.66666667,  -4.66666667, 35.33333333, 27.33333333,
             -35.66666667,  14.33333333],
           [ 30.33333333,  3.33333333, -26.66666667, 7.33333333,
             -39.66666667,   2.33333333]])
```

x. Consider a new array B below. How can one square every element of A, obtain the product of A^2B and sum all resulting values together, so that one is left with a single value 106584?

```
B = array([[0, 0, 1],
           [0, 0, 1],
           [0, 1, 1],
           [1, 1, 1],
           [1, 1, 1],
           [1, 1, 1]])
```

1. 2
2. 3×6
3. `A[1]`
4. `A[:,2]`
5. `A[1,4]`
6. `A[[1,1,2,2],[1,2,1,2]]`
7. `A.flatten()`
8. `A.reshape(9,2)`
9. `A - np.mean(A)`
10. `np.sum(np.matmul(np.power(A, 2), B))`

Question 7 [max 10 p]

Consider the toy dataset provided below, which is taken from SemEval-2018 Task 1: Affect in Tweets. The data consists of text strings representing users' Tweets and their associated valence labels. Here, the labels are integers ranging from -1, which represents a negative emotional state, to 1, which represents a positive emotional state. 0 is neutral. Given the data, provide `python` code for training a `LinearSVC` classifier in `scikit-learn` for valence classification. Your code should include three steps: featurization via `tf-idf`, training, and evaluation. Disregarding imports, your program should not exceed 10 lines. **NOTE:** since this dataset is very small, you should not expect to see a high accuracy.

Toy Dataset

```
X_train = [
    "I was going to say that Rooney is a shadow of his former self \
     but I don't want to offend any shadows. #NTFCvMUFC",
    "Left my phone in Mcds #panic",
    "It's 5:55am. I'm hungry but there is no food. #panic",
    "Charlotte and a friend just climbed to the top of a windmill! \
    Think the view was easy, breezy, beautiful? @PlayHollywoodU",
    "now that I have my future planned out, I feel so much happier \
    #goals #life  #igotthis #yay",
    "A #new day to #live and #smile. Hope all the #followers a nice \
    #night or #day. :D",
    "the ending of how I met your mother is dreadful",
    "#Sports Top seed Johnson chases double delight at \
     Tour Championship",
    "Howard Webb always held up as knowing the answers. No red. \
     Sutton and Craigan raging! Hahaha",
    "Well that was exhilarating. I didn't know you could have \
     goosebumps for 2 hrs 5m straight."
]

Y_train = [-1, -1, -1, 1, 1, 1, -1, 0, 0, 1]

X_test = [
    "The current run of superman continues to be a delight! \
     You can feel the love poured into it by everone on \
     the creative team #superman",
    "Turns out 'it' wasn't even anything to be concerned \
     about at all. Im happy and a bit frustrated it took \
     so long to get this answer.",
    "Don't join @BTCare they put the phone down on you, \
    talk over you and are rude. Taking money out of my \
    acc willynilly! #fuming",
    "Hey @gmail why can I only see 15 sent emails? \
    Where's the thousands gone?  #panic",
    "Congress attended by midterm liveliness differently \
    constraints: qmiv"
]

Y_test = [1, 0, -1, -1, 0]
```

```
vec = TfidfVectorizer(ngram_range=(1,3), analyzer="word")
X_train = vec.fit_transform(X_train)
svm = LinearSVC()
svm.fit(X_train, Y_train)
X_test = vec.transform(X_test)
print(svm.score(X_test, Y_test))
```