AlPro
Tp4 - Lists and trees using Python

1 Introduction

Linear and tree structures are essential to the organization of information. This tutorial is an opportunity to create and manipulate these data structures and to compare the execution times of the basic operations, i.e. adding, searching, deleting or displaying elements. In addition, you will also use the recursion principle to browse, modify or display your data structures. The theme of this tutorial is the organization of information related to persons through the management of a directory containing elements of type `Person`, itself containing information related to a person. This directory is represented with three different data structures: sorted list from python, sorted implemented linked list, binary search tree. All these data structures are ordered first according to the person's name, then his first name and finally his personal number (which will be chosen randomly, but we could take the social security number).

2 Data structures

To create a type `Person` we will use a *class* in the file `Types.py`:

```
class Person:
    ``description of the class''

    def __init__(self,name,fst_name,nbr):
        self.name=name
        self.fst_name=fst_name
        self.nbr=nbr

    def printPerson(self):
        print self.name, self.fst_name, self.nbr
```

This defines an object named `Person` and a constructor which can be called using for example:

```
pers=Person(``John'',''Doe'',20)
```

will initialize the variable pers to a new person named John Doe. It is called an *instance* of the class `Person`.

Think of it as a special kind of `struct` that have not only parameters (name, `fst_name` and `nbr`) but also functions like `printPerson`. Following the previous example, we can print `pers` using `pers.printPerson()`.

Now we will create new classes to build a linked list and a binary tree.

Question 1. For the linked list: in the file `Types.py` create a new class `PersonList` with a data which is of type `Person` and a link to a next `PersonList` as we did in C.

Question 2. For the binary tree: in the file `Types.py` create a new class `PersonNode` with a data of type `Person`, a left child and a right child.

Question 3. Add to each of these two classes a function to print an instance of the class (using the function `printPerson`).

Question 4. In a main.py file, build (by hand) a list and a tree with 3-4 people to test your printing functions.

3 Utilities

To be able to use the types and functions from the file Types.py inside a new file, you will need to import it. You can either use:

- **import** Types: you will need to write Types.function(...) every time you want to use a function.
- **from** Types **import** fct1, fct2: to import only the selected functions and then you are able to use fct1(...), fct2(...) without writing Types..
- **from** Types **import** *: to import all the content of Types without ever having to indicate Types before a function, type,... from Types.py.

Types, classes and functions behave the same way regarding the import. The same applies with standard library like sys, random, time,...

Define the following functions and implement them in a file Utilities.py:

Question 5. initArrayNamesFstname: Read the files Names.txt and First_names.txt and store the data into two lists that you will return at the end. (Yes, python allows you to build functions that return several elements, just using: **return** element1, element2, ...).

Question 6. generateFile: Random generation of a directory of 10000 people. This function will use the lists containing the first and last names, and a function gerenatePerson to create a person randomly (using a number between 0 and 10000000 for Person.nbr), then writes this person in the file People.txt.

Question 7. equalityPerson: Determines if two people have the same information: last name, first name and number. This function returns a boolean: true in case of equality, false in the other case.

Question 8. smallerPerson: Definition of an order on people (which will be used to order the data structures). This comparison is done first according to the name of the person, then his first name and finally his social security number. We consider that the two persons are not identical in the input of the function. This function returns a boolean: true if the first person is before the other, false in the other case.

Hint for the Q7 and Q8: string can be compared with the standard comparison operators (==,!=,<=,<,>=,>). Those who want to can replace these functions with comparison operators on Person using a (re)definition of the functions __eq__, __le__,... inside the class Person. This is called comparison operator overloading.

4 Creation and use of a file of people

Define the following functions and implement them in a file People.py for the three data structures: sorted list from python, sorted linked list and binary search tree. For implemented lists and trees, you need to make recursive functions. In the following questions you must replace “Structure” in the function name by “PythonList”, “List” and “Tree”.

Question 9. addStructure: Adding a person to the right place in the data structure using the comparison function. If a person is found to be identical to another one already present in the data structure, the person is not added. For linked lists and trees, these functions return the head of the new linked list or the root of the new tree, respectively.

Question 10. `readFileStructure`: Read information from the file "People.txt" to create the sorted data structure.

Question 11. `printStructure`: Printing of the data structure in the right order.

Question 12. `searchStructure`: Search for a person in the data structure by last name, first name and number. For the two lists structure, the function returns the index of the person found, otherwise -1. For the tree it returns the person, None otherwise.

Question 13. `removeStructure`: Deletion of a person in the data structure based on the name, first name and number. For the linked lists and the trees, you can use recursive functions.

5 Tests

Question 14. Compare the execution times of the three `readFileStructure` functions.

Question 15. Compare the execution times of the printing of the three structures.

Question 16. Compare the execution time of the search for 1000 persons in the three structures.

Question 17. Compare the execution times of deleting 1000 people in the three structures.

For the two last questions, you can build a list of 1000 people taken at random from the Person list of python. The execution times can be compared with the `time` library, in particular with the function `time.time()`.

What indicates the results of the timing tests ? Comment on the behavior of your functions in normal and borderline cases, for example when the input file is almost sorted or when it is completely mixed.

As an exercise, you may write on paper the algorithms for linked lists and trees for the questions 9 to 13.