Problem Set 1: Abstract Data Type – Sets and its applications

A set is an unordered collection of elements with no duplicates.

E.g. {1, 3, 8} is the same as {3, 8, 1}

Main methods in the Set ADT

- Set(): Creates a new set initialized to the empty set.
- Set(lyst): Creates a new set and initialized to the elements in the lyst. lyst is a list of items to be added to the set.
- length (): Returns the number of elements in the set. Accessed using the len() function.
- contains (element): Determines if the given value is an element of the set and returns the appropriate boolean value. Accessed using the in operator.
- add(element): Modifies the set by adding the given value or element to the set if the element is not already a member. If the element is not unique, no action is taken and the operation is skipped.
- remove(element): Removes the given value from the set if the value is contained in the set and raises an exception otherwise.
- equals (setB): Determines if the set is equal to another set and returns a boolean value. For two sets, A and B, to be equal, both A and B must contain the same number of elements and all elements in A must also be elements in B. If both sets are empty, the sets are equal. Access with == or !=.
- isSubsetOf(setB): Determines if the set is a subset of another set and returns a boolean value. For set A to be a subset of B, all elements in A must also be elements in B. Access with A <= B where statement is testing A is a subset of B
- properSubsetOf(setB): Determines if the set is a proper subset of setB and returns a Boolean value. For setA to be a proper subset of setB, setA must be a subset of setB, but setA is not equal to setB (i.e. there exist at least one element of B which is not an element of A). Access with A < B
- union(setB): Creates and returns a new set that is the union of this set and setB. The new set created from the union of two sets, A and B, contains all elements in A plus those elements in B that are not in A. Neither set A nor set B is modified by this operation. Access with A+B, where statement is executing A Union B
- intersect(setB): Creates and returns a new set that is the intersection of this set and setB. The intersection of sets A and B contains only those elements that are in both A and B. Neither set A nor set B is modified by this operation. Access with bitwise AND operator, e.g. A & B, where statement is executing A intersect B.
- Iterator: Implements the iterator, to allow items in the set to be iterated with a for loop in the client code.
- String Representation: Returns nicely printable string representation of the set.

Part A – Implementation of ADT

Implement the above ADT, implementing proper client program to test your code properly. Account for the performance of your ADT, due to be covered in Lesson 2.

Do also implement other functions like __str__ and appropriate iterator to allow for ease of use of your ADT.

Part B – Application of your ADT

Minimizing translations

Let's look at a problem, whose solution involves sets.

A European conference is being held in Brussels. The conference committee is designing signs to place outside the different meeting rooms to help delegates find the correct rooms. There is not enough space on a sign to use every language spoken at the conference. Fortunately, delegates are diplomats so most speak several different languages. The committee must find a small number of languages to put on the signs. These languages must be chosen so that every delegate can read at least one of them. Here's a more formal specification of this problem

Purpose: To find a small number of languages of which all delegates can read at least one

Input: The names of the languages spoken by each delegate. Data are in a text file called delegate.txt. Each line of the file contains the data for one delegate.

Output: A listing of languages of which all delegates can read at least one.

Illustration:

If there is some common group of languages spoken by all the diplomats, we can find it by taking the intersection of all the language sets. For example, here are the language sets for three diplomats:

{Dutch French German} {English French German Spanish} {French Italian German}

The intersection of these three sets is

{French German}

This intersection is the set of languages spoken by all three diplomats. Either of the languages in this intersection could be used alone for the signs.

Part 3: Challenge

It is, however, likely that there is no language spoken by all the diplomats attending the conference. The following three language sets illustrate this possibility:

{Dutch French German} {English French Spanish}{Italian German}

The intersection of these sets is the empty set; there is no language common to these three diplomats.

Upon inspection of these last three sets, we can see that if each sign is written in both French and German, all the diplomats can read it. Such inspections are easy with only a few diplomats. It is much more difficult when there are hundreds of diplomats speaking dozens of languages.

Part 3A:

> Write a program that solves the non-naïve scenario.

Part 3B:

> Your reflection: How does your algorithm perform given a larger number of languages and/or number of diplomats. Give an estimate of the performance given varying inputs.