14BHD COMPUTER SCIENCES, AA 2020/2021

Laboratory exercise 11

Goals of the exercise

- Data processing using the most common set operations
- Using a dictionary to search inside a table
- Use of complex structures for data storage

Technical contents

- Creation and use of sets and dictionaries
- Union, intersection and difference between sets
- Access to values and scan of elements of a dictionary

To be solved in the laboratory

~~*Exercise 1*~~. Write a program that counts the occurrences of each word in a text file. Next, improve the program so that it displays the most common 100 words. [P8.2] [P8.3]

~~*Exercise 2.*~~ Write a program that asks a user to type in two strings and that prints:

- The characters that occur in both strings;
- The characters that occur in one string but not the other;
- The letters that do not occur in either string.

Tip: Use *set* to turn a string into a set of characters. [P 8.9]

~~*Exercise 3.*~~ A sparse array is a sequence of numbers in which most entries are zero. An efficient way of storing a sparse array is a dictionary in which the keys are the positions with nonzero values, and the values are the corresponding values in the sequence. For example, the sequence **0 0 0 0 0 4 0 0 0 2 9** would be represented with the dictionary **{ 5:4, 9:2, 10:9 }**. Write a function **sparseArraySum**, whose arguments are two such dictionaries **a** and **b**, that produces a sparse array that is the *vector sum*; that is, the result's value at position *i* is the sum of the values of **a** and **b** at position *i*. [P8.22]

Exercise 4. Implement the *sieve* of *Eratosthenes*: a function for computing prime numbers, known to the ancient Greeks. Choose an integer **n**. This function will compute all prime numbers up to **n**.
First insert all numbers from 1 to n into a set. Then erase all multiples of 2 (***except 2***); that is, 4, 6, 8, 10, 12, ...
Erase all multiples of 3, that is, 6, 9, 12, 15, ...
Go up to **n**. The remaining numbers are all primes. [P8.4]

To be solved at home

Exercise 5. Write a "censor" program that first reads a file with "bad words" such as "**sex**", "drugs", "**C++**", and so on, places them in a set, and then reads an arbitrary text file. The program should write the text to a new text file, replacing each letter of any bad word in the text with an asterisk. [P8. 14]

Exercise 6. Write a program that reads the data from https://www.cia.gov/library/publications/the-world-factbook/rankorder/rawdata_2004.txt into a dictionary whose keys are country names and whose values are per capita incomes. Then the program should prompt the user to enter country names and print the corresponding values. Stop when the user enters *quit*. [P8.17]

Exercise 7. Write a program that reads a text file containing the image of maze, such as:

```
* *******
*   * * *
* ***** *
* * *   *
* * *** *
*   *   *
***** * *
*     * *
******* *
```

Here, the * are impassable walls and the spaces are corridors. Produce a dictionary whose keys are tuples (row, column) of corridor locations and whose values are sets of neighboring corridor locations. In the labyrinth example presented here, (1, 1) (blue square) has as adjacent corridors {(1, 2), (0, 1), (2, 1)}. Print the dictionary. [P 8.20]

Exercise 8. Continue the program from the previous exercise by finding an escape path from any point in the maze. Make a new dictionary whose keys are the corridor locations and whose values are the string "**?**". Then traverse the keys.

For any key that is at the boundary of the maze, replace the "**?**" with a value "**N**", "**E**", "**S**", "**W**", indicating the compass direction of the escape path. Now repeatedly traverse the keys whose values are "**?**" and check if their neighbors are not "**?**", using the first dictionary to locate the neighbors. Whenever you have found such a neighbor, replace the "**?**" with the compass direction to the neighbor.

Stop if you were unable to make any such replacement in a given traversal. Finally, print the maze, with the compass directions to the next escape location in each corridor location. For example,

```
*N*******
*NWW*?*S*
*N*****S*
*N*S*EES*
*N*S***S*
*NWW*EES*
*****N*S*
*EEEEN*S*
*******S*
```

[P 8.21]