DS 1st exam test, January 22nd, 2024

Download the data

1. Consider the file sciprog-qcb-2024-01-22-FIRSTNAME-LASTNAME-ID.zip and extract it on your desktop.

2. Rename sciprog-qcb-2024-01-22-FIRSTNAME-LASTNAME-ID folder:

   Replace **FIRSTNAME**, **LASTNAME**, and **ID** with your first name, last name and student id number. **Failure to comply with these instructions will result in the loss of 1 point on your grade.**

   like sciprog-qcb-2024-01-22-alessandro-romanel-432432

   From now on, you will be editing the files in that folder.

3. Edit the files following the instructions.

4. At the end of the exam, **compress** the folder in a zip file

   sciprog-qcb-2024-01-22-alessandro-romanel-432432.zip

   and submit it. This is what will be evaluated. Please, include in the zip archive all the files required to execute your implementations!

NOTE: You can only use the data structures and packages provided in the exam script files. **Importing other Python packages IS NOT allowed** unless explicitly stated in the exam instructions. Using Python collections or other libraries will impact your final grade. Still, **IT IS ALLOWED** to use **built-in Python operators** as we have done during the practical classes (max, min, len, reversed, list comprehensions, etc).

Exercise 1 [FIRST MODULE]

The dataset, titled "jobs_in_data.csv," offers comprehensive details about the salaries of workers worldwide in the field of Data Science. Each row in the table corresponds to an employee, and the columns include various information such as work_year, job_title, job_category, and more.

The dataset looks like the following:

	work_year	job_title	job_category	salary_currency	salary	salary_in_usd	employee_residence	experience_level	employment_type	work_setting	compar
0	2023	Data DevOps Engineer	Data Engineering	EUR	88000	95012	Germany	Mid-level	Full-time	Hybrid	
1	2023	Data Architect	Data Architecture and Modeling	USD	186000	186000	United States	Senior	Full-time	In-person	Ur
2	2023	Data Architect	Data Architecture and Modeling	USD	81800	81800	United States	Senior	Full-time	In-person	Ur

1) Load the dataset
2) Create the function **max_min_salary** to PRINT the "job_category" and "company_location" associated with the highest and lowest "salary_in_usd."

```
def max_min_salary(dataset):
    …
    return res
```

OUTPUT

```
> min salary:
>       Category:  Machine Learning and AI
>       Location: Germany
>       Salary:    14000
> MAX salary:
>       Category:  Data Science and Research
>       Location: United States
>       Salary:    450000
```

3) Develop a function, named "**get_exp_level**," which retrieves the "experience_level" of the employee holding the highest "salary_in_usd." Specify the associated "company_location" and "company_size."

   Test the function with "company_location" set to Ireland and "company_size" set to "M."

```
def get_exp_level(dataset, company_location, company_size):
    …
    return experience_level
```

4) Create a function that accepts dataset, "work_setting" (default value: "Remote") and "employee_residence" (default value: "Canada") as input parameters. The function should generate a dictionary with years as keys and nested dictionaries as values. The inner dictionary's keys should be "job_category," and the corresponding values should be the average and standard deviation of the salary. Finally, return and print the resulting dictionary.
NOTE: name the function **average_salary_over_time**

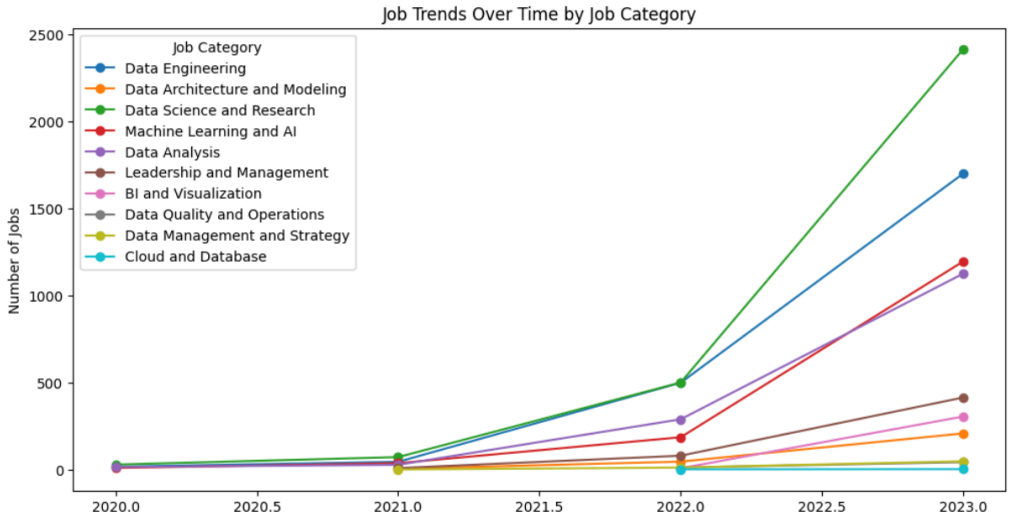
```
def average_salary_over_time(dataset, work_setting … ):
    …
    return dictionary
```

The resulting dictionary should have the following structure:

```
{
2023:
  {
    'Data Engineering': [salary mean, salary standard deviation],
    'Machine Learning and AI': [salary mean, salary standard deviation]
    ...
  },
2022:
  {
    'Machine Learning and AI': [salary mean, salary standard deviation],
    …
  },
…
}
```

5) Utilize the earlier function with the given parameters: employee_residence set to "United States" and work_setting set to "In-person." Save the resulting dictionary in a file named "res.json" using either the json module or the standard IO module.

6) Create a function named "plot_jobs_category_over_time(data)" that reproduces the following chart:



The chart depicts the evolution of job categories over time. Each line corresponds to a specific category, with the y-axis representing the number of employees and the x-axis denoting the years. Please ensure to include a legend, title, and label for the y-axis.

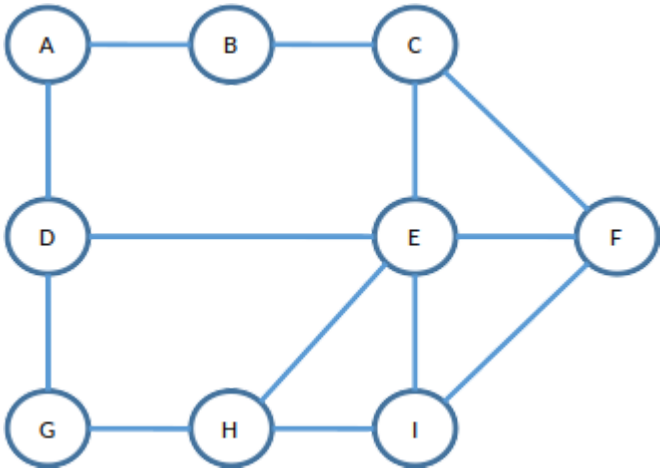
Exercise 2 [SECOND MODULE, theory]

Given a list *L* of *n* elements, a value *v*, a value *low* equal to 0 and value *high* equal to *n-1*, please compute the asymptotic computational complexity of the following function, explaining your reasoning.

```
def func(L, low, high, v):
  if high >= low:
    mid = (high + low) // 2
    if L[mid] == v:
      return mid
    elif L[mid] > v:
      return func(L, low, mid - 1, v)
    else:
      return func(L, mid + 1, high, v)
  else:
    return -1
```

Exercise 3 [SECOND MODULE, theory]

Describe the differences between the Depth-First and the Breadth-First Search algorithms for visiting graphs. Then, apply BFS to the graph below.

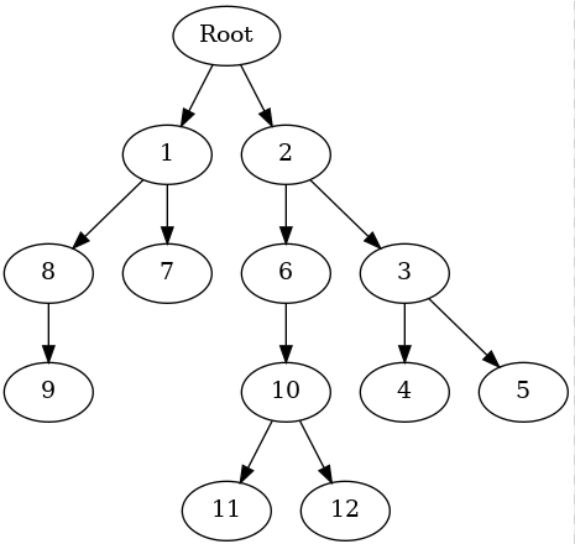


Exercise 4 [SECOND MODULE, practical]

Consider the BinaryTree implementation provided in the file exercise4.py. Complete the code where required:

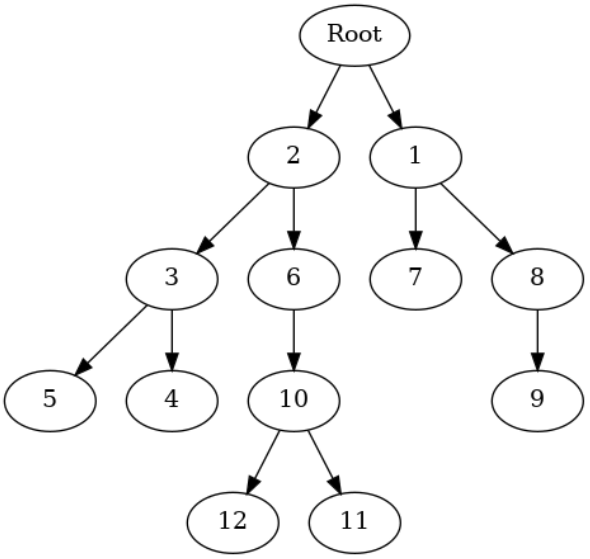
1. ```**mirror_binary_tree**``` is a function that takes as input a ```**BinaryTree**``` and mirrors the tree itself (each left and right nodes are mirrored).
Given the ```BinaryTree```:

```
Root: Root
    L-- 1
        L-- 8
            R-- 9
        R-- 7
    R-- 2
        L-- 6
            R-- 10
                L-- 11
                R-- 12
        R-- 3
            L-- 4
            R-- 5
```



you should obtain:

```
Root: Root
    L-- 2
        L-- 3
            L-- 5
            R-- 4
        R-- 6
            L-- 10
                L-- 12
                R-- 11
    R-- 1
        L-- 7
        R-- 8
            L-- 9
```



2. ``` **get_width_and_sum** ``` a function that takes as input a ```**BinaryTree**``` and returns a list of tuples, one for each level of the tree, containing three values:

- the level number (starting with root as 0);
- the width of the level (number of nodes in that level);
- the sum of the values of the nodes of that level.