Homework 4

DFA Simulation, Computation History, and Language Exploration

In this assignment, you will represent simple deterministic finite automata (DFAs) in Python, simulate their behavior, and produce small reports about their languages.

Goals:

- Track and display the computation history step by step
- Explore the language of a DFA up to a given length
- Collect statistics about state visits and transition usage

Starter Code

The following code is **given** to you as a skeleton.

You will complete the missing parts.

```
# DFA Example
dfa = {
   "states" : ["q0","q1"],
   "alphabet" : ["a","b"],
   "start_state" : "q0",
   "accepting_states" : ["q1"],
   "transitions" : {
      "q0":{"a":"q1",
          "b":"q0"},
      "q1":{"a":"q0",
          "b":"q1"}
   }
}

def simulate_DFA(dfa, word):
   current_state = dfa["start_state"]
   for char in word:
      current_state = dfa["transitions"][current_state][char]
   return current_state in dfa["accepting_states"]

def computation_history(dfa, word):
   initial_configuration = dfa["start_state"] + word
   pass

def language_exploration(dfa, n):
   pass
```

```
def language_report(dfa, n):
    pass
```

Rules

- You must implement:
  - `computation_history(dfa, word)`
  - `language_exploration(dfa, n)`
  - `language_report(dfa, n)`
- You may define additional **helper functions** if you want.
- Do **not** import any extra modules (no `itertools`, etc.). Use only basic Python.
- All tests will use strings over `dfa["alphabet"]` (no characters outside the alphabet).

Task 1 – `computation_history(dfa, word)`

This function should show how the DFA processes the input `word`,

returning a **list of configuration strings** representing each step.

Configuration Format

Each configuration is a string of the form:

```
<consumed_part>-<current_state>-<unconsumed_part>
```

- `consumed_part`: the prefix of the word that has already been read
- `current_state`: the DFA state after reading `consumed_part`
- `unconsumed_part`: the remaining suffix that has not been read yet

Start and End Configurations

- **Initial configuration** (before reading any symbol):

  ```
  -q0-ababa
  ```

  For example, when:
  - start state = `q0`
  - word = `"ababa"`

  Here:
  - consumed part: `""` (empty)
  - state: `q0`
  - unconsumed part: `"ababa"`

- **After each step**:
  - Read the next character
  - Update the state
  - Move that character from the unconsumed part to the consumed part
- **Final configuration** (after reading the entire word):

  ```
  ababa-q1-
  ```

  Here:
  - consumed part: `"ababa"`
  - state: `q1` (the final state after simulation)
  - unconsumed part: `""` (empty; nothing left to read)

Example

Given the DFA:

```
dfa = {
   "states" : ["q0","q1"],
   "alphabet" : ["a","b"],
   "start_state" : "q0",
   "accepting_states" : ["q1"],
   "transitions" : {
      "q0":{"a":"q1", "b":"q0"},
      "q1":{"a":"q0", "b":"q1"}
   }
}
```

and `word = "ababa"`.

The run of the DFA:

1. Start: state = `q0`, word = `"ababa"`
   - configuration: `"-q0-ababa"`
2. Read `a`: `q0 --a--> q1`
   - configuration: `"a-q1-baba"`
3. Read `b`: `q1 --b--> q1`
   - configuration: `"ab-q1-aba"`
4. Read `a`: `q1 --a--> q0`
   - configuration: `"aba-q0-ba"`
5. Read `b`: `q0 --b--> q0`
   - configuration: `"abab-q0-a"`

6. Read `a`: `q0 --a--> q1`
    - configuration: `"ababa-q1-"`

So:

```
computation_history(dfa, "ababa")
# should return:
# [
#   "-q0-ababa",
#   "a-q1-baba",
#   "ab-q1-aba",
#   "aba-q0-ba",
#   "abab-q0-a",
#   "ababa-q1-"
# ]
```

Expected Behavior

Some simple cases:

```
>>> computation_history(dfa, "")
["-q0-"]  # no characters, start and end are the same configuration

>>> computation_history(dfa, "a")
["-q0-a", "a-q1-"]
```

The function must:

- Return a **list**
- Each element of the list must be a **string** in the specified format

---

Task 2 – `language_exploration(dfa, n)`

This function explores the language of the DFA up to a given length `n`.

It should generate **all strings** over `dfa["alphabet"]` with length from `0` to `n`,

run the DFA on each one, and record whether it is accepted or rejected.

Details

- The alphabet is taken from:

  ```
  dfa["alphabet"]
  ```

  Example:

```
dfa["alphabet"] == ["a", "b"]
```

- You must generate all strings of lengths:

```
0, 1, 2, ..., n
```

- The length-0 string (epsilon) is represented as the **empty string** `""` in Python.

For each string:

- Call `simulate_DFA(dfa, word)`
- If it returns `True`, map the string to `"ACCEPT"`
- If it returns `False`, map the string to `"REJECT"`

Output Format

The function should return a dictionary, for example:

```
{
   "": "ACCEPT",
   "a": "REJECT",
   "b": "ACCEPT",
   "aa": "REJECT",
   "ab": "ACCEPT",
   ...
}
```

- **Key**: a generated string (Python `str`)
- **Value**: `"ACCEPT"` or `"REJECT"`

The exact order of keys is not important,

but using a consistent strategy (e.g., by length, then lexicographically) is recommended.

Example Usage

```
>>> language_exploration(dfa, 2)
{
   "": "REJECT",
   "a": "ACCEPT",
   "b": "REJECT",
   "aa": "REJECT",
   "ab": "ACCEPT",
   "ba": "ACCEPT",
   "bb": "REJECT"
}
```

```
# (This is just an example; actual results depend on the DFA.)
```

Task 3 – `language_report(dfa, n)`

This function also considers all strings of length `0` to `n`,

but instead of recording ACCEPT/REJECT per string, it produces a **statistics report**.

The report is a single dictionary that combines:

1. **State visit counts**
2. **Transition usage counts**
3. **Total accepted / rejected counts**

1. State Visit Counts

For each state, include an entry of the form:

```
"q0_visit_count": <how many times q0 was visited>,
"q1_visit_count": <how many times q1 was visited>,
...
```

Interpretation:

- Every time the DFA **enters** a state, this counts as a visit.
- The start state (`start_state`) is considered **visited once at the beginning** of each run.
- You must sum up all visits over **all strings** from length `0` to `n`.

2. Transition Usage Counts

For each transition, include an entry of the form:

```
"q0_a_q1_transition_count": <how many times q0 --a--> q1 was used>,
"q0_b_q0_transition_count": <how many times q0 --b--> q0 was used>,
"q1_a_q0_transition_count": <how many times q1 --a--> q0 was used>,
"q1_b_q1_transition_count": <how many times q1 --b--> q1 was used>,
...
```

The general key format is:

```
"<fromState>_<symbol>_<toState>_transition_count"
```

These counts should include all uses of each transition

over **all runs** on strings of length `0` to `n`.

3. Total Accepted / Rejected Counts

Add two more entries:

```
"total_accepted": <how many strings were accepted>,
"total_rejected": <how many strings were rejected>
```

Again, the counts should cover all strings with length from `0` to `n`.

Example Report

A possible output might look like this:

```
{
   "q0_visit_count": 37,
   "q1_visit_count": 25,
   "q0_a_q1_transition_count": 12,
   "q0_b_q0_transition_count": 20,
   "q1_a_q0_transition_count": 9,
   "q1_b_q1_transition_count": 21,
   "total_accepted": 10,
   "total_rejected": 22
}
```

The exact values will depend on the DFA and on `n`.

Submission Format

- Submit a single `.py` **file**.
  - Must use filename: `your_student_id.py`
- Your file must contain at least:
  - Your implementations of:
    - `computation_history(dfa, word)`
    - `language_exploration(dfa, n)`
    - `language_report(dfa, n)`
- Your code must run **without syntax errors**.

You may include some simple test calls at the bottom of the file

---

Grading

- **Correctness (90%)**
  - Functions return results in the required format
  - Functions behave correctly on different DFAs and inputs
- **Code Quality (10%)**

- Clear and meaningful variable names
- Short comments where helpful

Before submitting, try:

- Running `computation_history` on a few example words
- Running `language_exploration` with small `n` (like 2 or 3)
- Running `language_report` with small `n` and checking if the counts make sense

Example DFA Outputs (for testing your functions)

Aşağıdaki üç DFA, ödevdeki fonksiyonlar için **test örneği** olarak kullanılabilir:

- `computation_history(dfa, word)`
- `language_exploration(dfa, 3)`
- `language_report(dfa, 3)`

Her DFA için:

1. DFA tanımını veriyorum
2. Bazı `computation_history` örnekleri
3. `language_exploration(dfa, 3)` çıktısı
4. `language_report(dfa, 3)` çıktısı

DFA 1 – Even number of `a`'s

Language: all strings over `{a, b}` with an **even number of** `a`**'s**.

DFA Definition

```
dfa1 = {
    "states": ["q0", "q1"],
    "alphabet": ["a", "b"],
    "start_state": "q0",
    "accepting_states": ["q0"],
    "transitions": {
        "q0": {"a": "q1", "b": "q0"},
        "q1": {"a": "q0", "b": "q1"},
    }
}
```

`computation_history(dfa1, word)` Examples

```
computation_history(dfa1, "")
# ['-q0-']

computation_history(dfa1, "a")
# ['-q0-a',
#  'a-q1-']

computation_history(dfa1, "ab")
# ['-q0-ab',
#  'a-q1-b',
#  'ab-q1-']

computation_history(dfa1, "bab")
# ['-q0-bab',
#  'b-q0-ab',
#  'ba-q1-b',
#  'bab-q1-']
```

---

language_exploration(dfa1, 3)

All strings over `{a, b}` with length `0` to `3`:

```
language_exploration(dfa1, 3)
# {
#   "":   "ACCEPT",
#   "a":  "REJECT",
#   "b":  "ACCEPT",
#   "aa": "ACCEPT",
#   "ab": "REJECT",
#   "ba": "REJECT",
#   "bb": "ACCEPT",
#   "aaa": "REJECT",
#   "aab": "ACCEPT",
#   "aba": "ACCEPT",
#   "abb": "REJECT",
#   "baa": "ACCEPT",
#   "bab": "REJECT",
#   "bba": "REJECT",
#   "bbb": "ACCEPT"
# }
```

---

language_report(dfa1, 3)

```
language_report(dfa1, 3)
# {
```

```
#   "q0_visit_count": 32,
#   "q1_visit_count": 17,
#
#   "q0_a_q1_transition_count": 12,
#   "q0_b_q0_transition_count": 12,
#   "q1_a_q0_transition_count": 5,
#   "q1_b_q1_transition_count": 5,
#
#   "total_accepted": 8,
#   "total_rejected": 7
# }
```

DFA 2 – Strings ending with `"ab"`

Language: all strings over `{a, b}` that **end with** `"ab"`.

DFA Definition

```
dfa2 = {
   "states": ["q0", "q1", "q2"],
   "alphabet": ["a", "b"],
   "start_state": "q0",
   "accepting_states": ["q2"],
   "transitions": {
      "q0": {"a": "q1", "b": "q0"},
      "q1": {"a": "q1", "b": "q2"},
      "q2": {"a": "q1", "b": "q0"},
   }
}
```

`computation_history(dfa2, word)` Examples

```
computation_history(dfa2, "")
# ['-q0-']

computation_history(dfa2, "a")
# ['-q0-a',
#  'a-q1-']

computation_history(dfa2, "ab")
# ['-q0-ab',
#  'a-q1-b',
#  'ab-q2-']

computation_history(dfa2, "bab")
```

```
# ['-q0-bab',
#  'b-q0-ab',
#  'ba-q1-b',
#  'bab-q2-']
```

language_exploration(dfa2, 3)

```
language_exploration(dfa2, 3)
# {
#   "":   "REJECT",
#   "a":  "REJECT",
#   "b":  "REJECT",
#   "aa": "REJECT",
#   "ab": "ACCEPT",
#   "ba": "REJECT",
#   "bb": "REJECT",
#   "aaa": "REJECT",
#   "aab": "ACCEPT",
#   "aba": "REJECT",
#   "abb": "REJECT",
#   "baa": "REJECT",
#   "bab": "ACCEPT",
#   "bba": "REJECT",
#   "bbb": "REJECT"
# }
```

language_report(dfa2, 3)

```
language_report(dfa2, 3)
# {
#   "q0_visit_count": 27,
#   "q1_visit_count": 17,
#   "q2_visit_count": 5,
#
#   "q0_a_q1_transition_count": 11,
#   "q0_b_q0_transition_count": 11,
#   "q1_a_q1_transition_count": 5,
#   "q1_b_q2_transition_count": 5,
#   "q2_a_q1_transition_count": 1,
#   "q2_b_q0_transition_count": 1,
#
#   "total_accepted": 3,
#   "total_rejected": 12
```

```
# }
```

DFA 3 – At most one `a`

Language: all strings over `{a, b}` that contain **at most one** `a`

(0 or 1 `a` is OK, 2 or more `a`'s is rejected).

DFA Definition

```
dfa3 = {
   "states": ["q0", "q1", "q2"],
   "alphabet": ["a", "b"],
   "start_state": "q0",
   "accepting_states": ["q0", "q1"],
   "transitions": {
      "q0": {"a": "q1", "b": "q0"},
      "q1": {"a": "q2", "b": "q1"},
      "q2": {"a": "q2", "b": "q2"},
   }
}
```

`computation_history(dfa3, word)` Examples

```
computation_history(dfa3, "")
# ['-q0-']

computation_history(dfa3, "a")
# ['-q0-a',
#  'a-q1-']

computation_history(dfa3, "ab")
# ['-q0-ab',
#  'a-q1-b',
#  'ab-q1-']

computation_history(dfa3, "bab")
# ['-q0-bab',
#  'b-q0-ab',
#  'ba-q1-b',
#  'bab-q1-']
```

`language_exploration(dfa3, 3)`

```
language_exploration(dfa3, 3)
# {
#   "":   "ACCEPT",
#   "a":  "ACCEPT",
#   "b":  "ACCEPT",
#   "aa": "REJECT",
#   "ab": "ACCEPT",
#   "ba": "ACCEPT",
#   "bb": "ACCEPT",
#   "aaa": "REJECT",
#   "aab": "REJECT",
#   "aba": "REJECT",
#   "abb": "ACCEPT",
#   "baa": "REJECT",
#   "bab": "ACCEPT",
#   "bba": "ACCEPT",
#   "bbb": "ACCEPT"
# }
```

---

language_report(dfa3, 3)

```
language_report(dfa3, 3)
# {
#   "q0_visit_count": 26,
#   "q1_visit_count": 16,
#   "q2_visit_count": 7,
#
#   "q0_a_q1_transition_count": 11,
#   "q0_b_q0_transition_count": 11,
#   "q1_a_q2_transition_count": 5,
#   "q1_b_q1_transition_count": 5,
#   "q2_a_q2_transition_count": 1,
#   "q2_b_q2_transition_count": 1,
#
#   "total_accepted": 10,
#   "total_rejected": 5
# }
```