MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CENG 111

Fall 2023

Take-Home Exam 2

REGULATIONS

Due date: 23:59, 4 December 2023, Sunday *(Not subject to postpone)*

Submission: Electronically. You should save your program source code as a text file named `the2.py`. Check the announcement on the ODTUCLASS course page for the submission procedure.

Team: There is **no** teaming up. This is an EXAM.

Cheating: Source(s) and Receiver(s) will receive zero and be subject to disciplinary action.

INTRODUCTION

A common issue with working families with babies is organizing a schedule for babies' care: Who will look after the babies and when? In this THE, your task will be to help a family to check whether a proposed schedule is feasible or not.

PROBLEM & SPECIFICATIONS

Assume that, in this family, there is only one baby and that there are various *agents* that can take care of the baby: The "mother", "father", "babysitter", "grandma", "aunt", "other aunt", "neighbour lady". We will encode them with `"m","f","b","g","a1","a2","n"`. There are certain ground rules associated with these individuals:

$RULE_1$ **mother (`m`):** She works 4 days a week. But the off-day has to be fixed all over the month.
[Each day of her care-taking has a cost of 10 units]

$RULE_2$ **father (`f`):** Works on all workdays of the week. Can take two workdays off per month. These days cannot be consecutive.
[Each day of his care-taking has a cost of 20 units]

$RULE_3$ **babysitter (`b`):** She is always available. However, if she is booked on two consecutive days that are less than 3 days (whether or not they are weekdays or weekend days) apart (e.g., Monday and Tuesday are 0-day apart whereas Friday and the following Monday are two days apart), she will also charge for the intervening days, even though she won't be babysitting on those days.
[Each day of her care-taking has a cost of 30 units]

$RULE_4$ **grandma (`g`):** Is usually available. She plays cards every Wednesday. Her card-play group has a strict rule that any group member can miss only one meeting per month.
[Each day of her care-taking has a cost of 50 units]

$RULE_5$ **aunt (`a1`):** Only available on Tuesdays and Fridays.
[**Each day of her care-taking has a cost of 32 units**]

$RULE_6$ **other aunt (`a2`):** Available on all days but has a grudge against aunt `a1`. Refuses to enter the house the following day aunt `a1` has babysat. She says "I can still smell her presence!"
[**Each day of her care-taking has a cost of 27 units**]

$RULE_7$ **neighbour lady (`n`):** Available on Monday, Tuesday and Wednesday.
[**The first babysitting in the month does not cost anything. The successive ones (in that month) are power of 5. So the first is 0 units, the second is 5, the third is 25, and so on.**]

Your solution should conform to the following requirements:

- The baby does not need to be babysat at the weekends. Therefore you will be given a sequence of 20-22 days (weekdays of a month) of babysitting assignments as a list as follows:

  ```
  ["g","m","m","f","a1","a2",...,"b"]
  ```

  We name this list a *month-calendar*.

- You will be evaluating the validity of the month-calendar as far as the rules (defined above) are concerned. While doing so, the "consecutive" and "following" relations between days should include weekends into account (i.e., Saturday and Friday are consecutive).

- A **valid** month-calendar is the one that breaks no rules. If a month-calendar is valid, it is evaluated for the cost.

- The month-calendar starts always with Monday, followed by the next weekday, and so on. After Friday, the list continues with Monday.

- Your task is to write a function `check_month(month_calendar)`.

  - It shall take a single argument, a month-calendar, and return a list of breached rule numbers if one or more rules are breached. The order of the rule numbers in the list is not important.
  - If no breach is discovered, an integer of the whole month's babysitting cost (in units) is returned.

EXAMPLE RUN

The following illustrates how the function `check_month(month_calendar)` will be used:

```
>>> M1 = [
    "m", "b", "g", "b", "f",
    "m", "a1", "n", "a2", "f",
    "m", "b", "f", "a1", "b",
    "m", "b", "a1", "a2", "n"
]
>>> check_month(M1)
[2, 5, 6, 7]
```

```
>>> M2 = [
    "g","m","f","a2","a1",
    "b","n","n","g","a1",
    "n","m","b","b","b",
    "b","m","f","s","a2",
    "a2"
]
>>> check_month(M2)
550
```

RESTRICTIONS and GRADING

- You are strictly forbidden from using Python's repetitive statements (`for` and `while`). However, you are allowed to use list comprehension and higher-order functions operating on lists. `zip`, `enumerate` are functions that come in handy.
- A set of arbitrarily selected students will be subject to oral examination about their solutions. The details will be announced on ODTUclass later.
- Your program will be graded through an automated process and therefore, any violation of the specifications will lead to errors (and reduction of points) in automated evaluation. You should especially avoid printing something on screen.
- Your solutions will not be tested with incorrect/erroneous inputs.
- Your program will be tested with multiple data (a distinct run for each data). Any program that performs only 30% and below will enter a glass-box test (eye inspection by the grader TA). The TA will judge an overall THE2 grade in the range of [0,30].
- A program based on randomness will be graded zero.
- The glass-box test grade is not open to discussion nor explanation.