Święta Bajtoru

# Opis zadania

W królestwie Bajtor daty świąt państwowych są ruchome. Corocznym zadaniem króla Bajtoru jest wyznaczenie terminów wszystkich świąt. Po wielu latach swojego panowania król zauważył, że większość Bajtorianów zamiast celebrować dane święto, woli wykorzystać dzień wolny na odpoczynek. Co więcej, badania wykazały, że poparcie dla danego święta jest najwyższe, kiedy święto przypada na **piątek lub poniedziałek**. Analitycy uważają, że istnieje znacząca korelacja między spostrzeżeniem króla, a wnioskami z badań – otóż wielu Bajtorianów korzysta wtedy z tak zwanych długich weekendów.

Król Bajtoru jest monarchą, który chce zapewnić swoim obywatelom jak największe zadowolenie, więc w tym roku postanowił, że podczas wyznaczania dat świąt weźmie pod uwagę tę zależność i **co najmniej**  świąt państwowych wypadnie w piątek lub poniedziałek.

Jednak tutaj napotkał problem, gdyż kalendarze na **lata 2023 i 2024** jeszcze nie powstały, więc nie może sprawdzić jaka data odpowiada któremu dniu tygodnia. Król Bajtoru stwierdził, że przekaże ci propozycje terminów świąt, a ty sprawdzisz ile procent z podanych dat przypada w piątek lub poniedziałek w 2023 i 2024 roku. Dodatkowo prosi cię o podanie minimalnej ilości dat, musi zmienić, aby co najmniej 50% świąt przypadało w piątek lub poniedziałek. Jedyne informacje jakie posiadasz, to że  **to niedziela i to poniedziałek**.

Uwaga! Pamiętaj, że w kalendarzu gregoriańskim, którym posługuje się królestwo Bajtor, rok jest przestępny, czyli luty ma .

# Wejście – specyfikacja pliku wejściowego

* w pierwszej linii znajduje się liczba () określająca ilość pozostałych linii do wczytania
* każda kolejna linia zawiera jedną datę w formacie
  + i to liczby składające się z dwóch cyfr każda; liczba ta symbolizuje odpowiednio dzień i miesiąc
  + liczby te mogą zawierać poprzedzające zero, na przykład 01.01
  + przykłady:
    - oznacza 4. dzień marca
    - oznacza 20. dzień maja
  + Uwaga: jest zapewnione, że podana data jest poprawna oraz w pliku nie wystąpi data

# Wyjście – specyfikacja pliku wyjściowego

W pliku „”:

* w 1. wierszu wynik dla ; w 2. wierszu wynik dla
* format wyniku: procent dat z propozycji króla, które w danym rokuwypadają w piątek lub poniedziałek oraz po spacji minimalna ilość terminów, które król musi zmienić, aby spełnić warunek co najmniej świąt przypadających w piątek lub poniedziałek dla danego roku
* procent należy podać z dokładnością do dwóch miejsc po przecinku
* jeśli król nie musi wprowadzać zmian w propozycjach dla danego roku, to należy podać procent i po spacji „”

# Przykład

Załóżmy, że król podaje propozycje terminów świąt. Po wykonaniu obliczeń, okazuje się, że dla roku tylko z nich przypadają w poniedziałek lub piątek, a dla roku terminów przypada w poniedziałek lub piątek. W takiej sytuacji plik wyjściowy powinien zawierać odpowiedzi:

* w 1. wierszu:
* w 2. wierszu:

# Rozwiązanie, program

def prefixSums(nums: []): # prefix sums for optimization of computational complexity  
 sums = [0]  
 for index in range(1, len(nums)):  
 sums.append(sums[index - 1] + nums[index - 1])  
 print("sums:", sums)  
 return sums  
  
  
def getDatesToChange():  
 if percentOfFridayOrMonday < 0.5:  
 if datesQuantity % 2 == 1:  
 return (datesQuantity // 2) + 1 - fridayOrMondayCounter  
 else:  
 return (datesQuantity // 2) - fridayOrMondayCounter  
 else:  
 return 0  
  
  
def writeAnswer():  
 print(fridayOrMondayCounter, "/", datesQuantity)  
 print(percentOfFridayOrMonday, datesToChange)  
 outputFile.write(str(percentOfFridayOrMonday) + "% " + str(datesToChange) + "\n")  
  
  
outputFile = open("wyniki.txt", "w")  
inputFile = open("daty.txt", "r")  
lines = inputFile.readlines()  
inputFile.close()  
  
datesQuantity = int(lines[0])  
lines.pop(0)  
dates = list(map(lambda line: line.strip(), lines))  
  
print("datesQuantity:", datesQuantity)  
print("dates:", dates)  
  
# --- 2023 ---  
  
# number or days in the following months of 2023 year (regular year)  
months = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]  
monthsDaysSums = prefixSums(months)  
  
fridayOrMondayCounter = 0  
for date in dates:  
 day = int(date[:2])  
 month = int(date[-2:])  
  
 totalDays = monthsDaysSums[month - 1] + day - 1  
 dayOfWeek2023 = (totalDays % 7) # 0 -> Sunday, 1 -> Monday, etc., 5 -> Friday  
 print(date, totalDays, dayOfWeek2023)  
  
 if dayOfWeek2023 in [1, 5]:  
 fridayOrMondayCounter += 1  
  
percentOfFridayOrMonday = fridayOrMondayCounter / datesQuantity  
datesToChange = getDatesToChange()  
print("datesToChange:", datesToChange)  
  
percentOfFridayOrMonday = round(percentOfFridayOrMonday \* 100, 2)  
writeAnswer()  
  
# --- 2024 ---  
  
# number or days in the following months of 2024 year (leap year - February has 29 days)  
months = [31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]  
monthsDaysSums = prefixSums(months)  
  
fridayOrMondayCounter = 0  
for date in dates:  
 day = int(date[:2])  
 month = int(date[-2:])  
  
 totalDays = monthsDaysSums[month - 1] + day - 1  
 dayOfWeek2024 = (totalDays % 7) # 0 -> Monday, 1 -> Tuesday, etc., 4 -> Friday  
 print(date, totalDays, dayOfWeek2024)  
  
 if dayOfWeek2024 in [0, 4]:  
 fridayOrMondayCounter += 1  
  
percentOfFridayOrMonday = fridayOrMondayCounter / datesQuantity  
datesToChange = getDatesToChange()  
print("datesToChange:", datesToChange)  
  
percentOfFridayOrMonday = round(percentOfFridayOrMonday \* 100, 2)  
writeAnswer()  
outputFile.close()