Algorithms and Data Structures SoSe 25 Solutions

Igor Dimitrov

2024-12-18

Table of contents

Preface

1 Sheet 1

1.1 Friday the 13th

Every now and then friday coincides with the 13th day of a month. How often does this happen?

First we define a function to determine whether a year is a leap year:

```python
def is_leap(year) -> bool :
    """output true iff year is a leap year
    precondition: year >= 0

    >>> is_leap(1999)
    False
    >>> is_leap(2000)
    True
    >>> is_leap(1900)
    False
    """
    return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)

print(
    is_leap(2000),
    is_leap(1999),
    is_leap(1900))
```

```
True False False
```

Next we write a function that converts the day of a year to the day of the month that this day would fall on. This function takes leap years into account:

```python
def convert_day(year_day, isLeap = False) :
    """convert a year day to a month day
```

```
    precondition: year_day in [1...365] for default call
    precondition: year_day in [1...366] for leap year call, with isLeap = True
    for a given year day year_day returns the day of the month the day falls on
    taking into account if a year is a leap year with the isLeap parameter

    >>> convert_day(1)
    1 # 1st day of a year is the 1st day of January
    >>> convert_day(255)
    12 # 255th day of a year is the 12th day of August
    >>> convert_day(60, True)
    29 # 60th day of a leap year is the 29th day of February
    >>> convert_day(366, True)
    31 # the last of a leap year is the 31th December
    """

    # Days in each month (non-leap year)
    months = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    if isLeap : months[1] = 29

    i = 0 # month number starting with 0
    s = 0 # accumulates the sum of the days in months
    # invariant: s == sum(months[0..i-1]) and day > s
    while year_day > s + months[i] :
        s += months[i]
        i += 1
    # sum(months[0..i-1]) == s < day <= s + months[i] == sum(months[0..i])
    return year_day - s
```

We test the function for a few days, including

- 1st day of a non-leap year: January 1st
- 255th day of a non-leap year: September 12th
- 366th day of a leap year: December 31st
- 60th day of a leap year: February 29th

```
print(
    convert_day(1),
```

5

```
    convert_day(255),
    convert_day(366, True),
    convert_day(60, True), sep = '\n'
)
```

```
1
12
31
29
```

Next we compute the days of the year that fall on the 13th day of a month, both for a leap and a non-leap year.
We will use this information in the next function.

```
thirteenth_days = []
for i in range(1, 366) :
    if convert_day(i) == 13 : thirteenth_days.append(i)

thirteenth_days

thirteenth_days_leap = []
for i in range(1, 367) :
    if convert_day(i, True) == 13 : thirteenth_days_leap.append(i)

print(thirteenth_days,
    thirteenth_days_leap, sep = '\n')
```

```
[13, 44, 72, 103, 133, 164, 194, 225, 256, 286, 317, 347]
[13, 44, 73, 104, 134, 165, 195, 226, 257, 287, 318, 348]
```

Now we define a function that accepts a parameter that stands for the day of the week, ranging from 0 to 6, representing the days Mo, …, Sun.

This input parameter is the day of the week that the first 13th day of the month (i.e. January 13th) fell on.
E.g. if January 13th was Wednesday that year, then the input parameter is 2. If the year is a leap year a second default boolean parameter is provided as `True`.

```python
def days_of_the_week_that_are_thirteenth(i, isLeap = False) :
    """return an array that contains the days of the week
    that are the thirteenth day of the month

    days are numbered from 0 to 6 corresponding to days Mon .. Sun
    input parameter i is the number of the first 13th day. It can
    be any day of the week. E.g. if in a given year Januaray 13th was Monday
    then i is 0.

    """

    thirteenth_days = [13, 44, 72, 103, 133, 164, 194, 225, 256, 286, 317, 347]
    if isLeap :
        thirteenth_days = [13, 44, 73, 104, 134, 165, 195, 226, 257, 287, 318, 348]


    b = [None] * len(thirteenth_days)
    b[0] = i
    for j in range(1, len(thirteenth_days)) :
        b[j] = (b[j - 1] + (thirteenth_days[j] - thirteenth_days[j - 1])) % 7
    return b


print(days_of_the_week_that_are_thirteenth(1),
    days_of_the_week_that_are_thirteenth(3, True), sep = '\n')
```

```
[1, 4, 4, 0, 2, 5, 0, 3, 6, 1, 4, 6]
[3, 6, 0, 3, 5, 1, 3, 6, 2, 4, 0, 2]
```

Note that January 13th can be any day of the week from Monday to Sunday for a given year. This function computes the days of the week all the 13th days of a month fall on given the day of the week 13th January falls on that year. E.g. for the year 1993, January 13th was a Wednesday. So the input parameter is simply the number 2.

All the other 13th days of other months are computed that as follows with this function:

```python
print(days_of_the_week_that_are_thirteenth(2, is_leap(1993)))
```

```
[2, 5, 5, 1, 3, 6, 1, 4, 0, 2, 5, 0]
```

From this result we see that the year 1993 had only one Friday the 13th - on August.

Next we define another function that computes the first day of any year after (and including) 1 A.D.

It is assumed that the first day of the first year was a Monday. The first years of all other years are computed accordingly and taking leap years into account:

```python
def first_day_of_year(year) :
    """return the first day of a year given as input parameter
    where days are numbered 0..6 starting from monday
    precondition: year >= 1


    >>> first_day_of_year(1999)

    """
    days_past = 0
    for i in range(1, year) :
        if is_leap(i) : days_past += 366
        else : days_past += 365
    return days_past % 7
```

Let's test this function to find out the what day of the week the first days of the first 5 years fell on:

```python
for i in range(1, 6) : print(first_day_of_year(i))
```

```
0
1
2
3
5
```

First day of year 1993:

```python
first_day_of_year(1993)
```

```
4
```

Now we are ready to generate all the 13th days of all months of years ranging from 1993 to 2025.

```
thirteenth_days = []
for y in range(1993, 2026) :
    first_day = first_day_of_year(y)
    first_thirteenth_day = (first_day + 12) % 7
    thirteenth_days.append(days_of_the_week_that_are_thirteenth(first_thirteenth_day, is_leap
```

`thirteenth_days` holds 33 arrays, each holding 12 values, that contains the days of the week that the 13th of the corresponding month fell on:

```
print(thirteenth_days)
```

To find out how many times 'Friday the 13th' was experienced since 1993, we simply have to count and add iup each occurrence of '4' (Friday) in each array:

```
count = 0
for days in thirteenth_days :
    for day in days :
        if day == 4 : count += 1

print(count)
```

55

So Friday the 13th was experienced 55 times in total since year 1993. Finally, we can encapsulate all of this computation in a single function:

```
def count_friday_13th_since(year) :
    thirteenth_days = []
    for y in range(year, 2026) :
        first_day = first_day_of_year(y)
        first_thirteenth_day = (first_day + 12) % 7
        thirteenth_days.append(
            days_of_the_week_that_are_thirteenth(first_thirteenth_day, is_leap(y)))
    count = 0
    for days in thirteenth_days :
        for day in days :
            if day == 4 : count += 1
```

```
    return count


count_friday_13th_since(2000)
```

44

1.2 Stable Glasses

a) *Linear search*: **30** tries in worst case, corresponding to the situation when the glass breaks at the highest platform. We start at the bottom platfrom and move to the next higher one if the glass doesn't break. As soon as the glass breaks we know that the platform just below is the maximum height the glass can withstand..

b) *Binary search*: **5** tries always. In worst case each time we try the glass breaks. Therefore, maximum amount of broken glasses is **5**. This corresponds to the situation where no height is safe for the glass.

   Justification: The general strategy is the same as with binary search - we first try the middle platform, i.e. 15th platform. In case the glass doesn't break we know that we must search only in the upper half. Otherwise, we must search only in the lower half. Since in each step we half the amount of platforms the procedure is guaranteed to terminate in approximately $\log_2 30$ steps. How many exactly can be seen with the following sequence: 15, 8, 4, 2, 1. Explanation of this sequence is that the middle platform where we have to perform the experiment is determined by the formula: `(total platforms left + 1) // 2`. And the total platforms left in the next step is always the number of the middle platform from the previous step. So in total there will be 5 steps and each time a glass will break.

c) In this case we can determine the highest platform in **11** steps at worst case with the following procedure (by dividing the stand in 3 equal parts)

   1. Test first glass in the highest platform in the lower third, i.e. the 11th platform. If it breaks then we must test the other glass in the remaining 10 platforms in the lower third linearly until it breaks the first time: $10 + 1 = \mathbf{11}$ tries at worst.
   2. if it doesn't break, then we have to try the lowest platform in the upper third, i.e. the 21st platform. If the glass breaks here, then we have to search in the middle third, i.e. we have to test the platforms 12..20 with the other glass linearly, until it breaks. This totals to $1 + 1 + 9 = \mathbf{11}$ tries in worst case.
   3. If the glass doesn't break in the 21st platform, then we must search the upper third platform with the other glass, namely the platforms 22...30, linearly until the glass breaks. This totals to $1 + 1 + 9 = \mathbf{11}$ tries in worst case.