Zadanie Dostawa – LOGIA 19 SP (2018/19), etap 3

Treść zadania

Fabryka produkuje 3 typy ozdób: *a*, *b* i *c*. Klient określa zamówienie podając kod składający się z liter *a*, *b*, *c*. Kod opisuje, ile ozdób danego rodzaju wchodzi w skład zamówienia. Na przykład kod *baabca*, oznacza, że w skład zamówienia wchodzą trzy ozdoby *a*, dwie *b* i jedna *c*. Zamówienia są realizowane w kolejności zgłoszenia. Jeśli brakuje danego rodzaju ozdoby do zrealizowania zamówienia, to całe zamówienie zostaje anulowane.

Napisz dwuparametrową funkcję **dostawa**, której pierwszym parametrem jest niepusta lista słów reprezentujących zamówienia w kolejności zgłoszeń, a drugim liczba nie mniejsza niż 1 i nie większa niż 250 określająca początkowy zapas każdego rodzaju ozdób. Wynikiem funkcji jest liczba zrealizowanych zamówień.

Przykłady:

Wynikiem **dostawa(["abac", "baaba", "acac", "babc"], 6)** jest **3**

Nie może być zrealizowane trzecie zamówienie, ponieważ zabraknie ozdób typu *a*.

Wynikiem **dostawa(["abc","abaa", "caa", "bbbcc", "cac","abc"], 5)** jest **4**

Nie może być zrealizowane zamówienie trzecie (zabraknie *a*) i szóste.

Omówienie rozwiązania

Rozwiązanie warto podzielić na dwie części: analizowanie poszczególnych słów opisujących konkretne zlecenia oraz przeglądanie listy zamówień i wybieranie tych, które można zrealizować. Zauważmy, że zamówienia są rozpatrywane w kolejności zgłoszenia, a te z nich, których nie można w danej chwili zrealizować odrzucamy. Innymi słowy, nie staramy się zrealizować jak największej liczby zamówień, wyszukując je odpowiednio na liście, lecz przeglądamy listę po kolei.

Dla każdego zamówienia na liście należy policzyć trzy wartości: liczbę ozdób *a*, *b* oraz *c*. Każde z nich wymaga obejrzenia całego kodu zamówienia litera po literze. Gdy w badanym słowie występuje interesująca nas litera zwiększamy odpowiedni licznik. Zamiast pisać trzy osobne funkcje dla trzech typów ozdób, można przygotować jedną funkcję **ile_liter()**. Jej wynikiem będzie lista złożona z trzech liczb, odpowiadających elementom *a*, *b* oraz *c*.

Po przygotowaniu funkcji do szacowania zapasów ozdób niezbędnych do realizacji konkretnego zamówienia, możemy skupić się na zaplanowaniu całej dostawy. Przeglądamy kolejne zamówienia na liście, dla każdego z nich sprawdzając, czy może zostać zrealizowane. Jeśli tak, to zmniejszamy odpowiednio zapasy ozdób i zwiększamy licznik zrealizowanych dostaw.

Rozwiązanie w języku Python

```python
def ile_liter(wyraz):
    wynik = []
    # zerujemy liczniki
    ilea = ileb = ilec = 0
    # zliczamy poszczególne litery w wyrazie
    for litera in wyraz:
        if litera == "a":
            ilea = ilea+1
        if litera == "b":
            ileb = ileb+1
        if litera == "c":
            ilec = ilec+1
    # budujemy listę złożoną z liczby liter a, b i c
    wynik.append(ilea)
    wynik.append(ileb)
    wynik.append(ilec)
    return wynik

def dostawa(lista, zapas):
    ilez = 0
    # ustawiamy początkowe wartości zapasu produktów
    zapas_a = zapas_b = zapas_c = zapas
    # przeglądamy poszczególne zamówienia
    for wyraz in lista:
        # zliczamy produkty w zamówieniu
        w = ile_liter(wyraz)
        # sprawdzamy, czy starczy  zapasu na dane zamówienie
        # jeśli tak, to realizujemy i zwiększamy licznik zamówień
        if (zapas_a >= w[0]) and (zapas_b >= w[1]) and (zapas_c >= w[2]):
            zapas_a = zapas_a - w[0]
            zapas_b = zapas_b - w[1]
            zapas_c = zapas_c - w[2]
            ilez = ilez + 1
    return ilez

```

Testy

Testowanie zaczynamy od przykładów podanych w treści zadania. Następnie dobierając odpowiednio początkowe wartości zapasów oraz listę zamówień staramy się sprawdzić różne możliwe przypadki: gdy można zrealizować wszystkie zlecenia, ani jednego, dokładnie jedno, dwa, trzy itp.

Poniższe testy obejmują proste przypadki, które można bez trudu przeanalizować ręcznie oraz trudniejsze, które obejmują również przypadki wygenerowane losowo.

Wywołanie – Python	**Wynik**
dostawa(["aaa", "bbb", "ccc"], 3)	3
dostawa(["baaacaaa", "bbb", "bbccbbb"], 5)	1
dostawa(["b", "bbb", "bbbbbbb"], 15)	3
dostawa(["baba", "baba", "baba"], 1)	0
dostawa(["baaacaaa", "bbccbbb", "bbb"], 5)	1
dostawa(["aacbacaaccbbbbbcbccbacaababcca", "bcaccaaabcacbbbbacbbabbacaaccc", "abacaabacccabbbbcaaccccbbcbaab", "acabccbacaaabbccaaaabccbbbbbcc", "ccbcbabbabacbacacbcaabcbaacacb", "baabcccabbbbccbaaabcabcabaaccc", "cbabcbcabccabbabccaccaacbbbaaa", "bbacbbcbcbaacbcccabcbabaaaccaa", "babbcaabbaaccacabcabcccbacbabc", "abbcaaabaabbcbbccccbcacacbaacb"], 50)	5
dostawa(["cbacaaccbbbbbcbccbacaa", "ccaaabcacbbbbacbbabbacaac", "bacccabbbbcaaccccb", "acaaabbccaaaabccbbbb", "abbabacbacacbcaabcbaacac", "cabbbbccbaaabcabcabaac", "bcbcabccabbabccaccaacbbb", "cbcbaacbcccabcbab", "baaccacabcabcccbacb", "bcaaabaabbcbbccccbcacacbaac"], 30)	4
dostawa(["acca", "cccab", "cbabacbc", "caabcbca", "baca", "acabba", "bcbc", "abc", "ac", "bcca", "ccbaabc", "bacb", "bbac", "caabcb", "bbcc", "abacbab", "abbca", "ab", "cbbc", "cbbaba", "abc", "ccba", "a", "abcaabc", "cabccaba", "aabcca", "ccb", "abaab", "ccabac", "bcbb", "bbbacc", "c", "acba", "baccaabb", "bacac", "bba", "cccb", "abcaa", "bcbaa", "cabbcc", "accacabbb", "caccbb", "caa", "cbcb", "acccb", "caba", "cccaa", "bcabc", "aaab", "cbab", "bb", "ac", "cacbb", "cabacb", "acca", "bcaaac", "ccab", "babca", "acacbbb", "bab", "cbbaacc", "cbbc", "abbccca", "abbac", "cbbc", "caabcacbb", "c", "abacbca", "caaabcb", "bcbac", "bccbacba", "bcca", "bbcaccaa", "accb", "cbbbcc", "a", "bacaa", "cabca", "cca", "accbc", "bbac", "abaccbc", "bccaa", "bcc", "acbbca", "ccabaabcb", "baca", "bbaacacc", "bac", "acabcac", "baaa", "ababccc", "aabab", "ccbcab", "cacca", "cabaabc", "bccacaab", "cabbb", "acbc", "cabcacba"], 100)	64

dostawa(["acbcba", "bbaaacb", "accbb", "abccabac", "ac", "ccbbaacab", "baba", "baaabc", "baac", "baccab", "bacc", "bbacc", "abcab", "abbb", "bcaa", "c", "cbabab", "bbccbca", "abac", "aacbb", "cacabcab", "cabab", "aabbccac", "accbbba", "caaccb", "abac", "bacacb", "acabb", "bccbca", "bb", "abccb", "aaaccbbc", "bbbc", "ba", "aaa", "caacabcb", "bbcab", "baacba", "acaab", "bcab", "cbbaab", "bbaa", "bcbba", "cba", "cbbcaa", "abaa", "cacc", "bccac", "acacb", "ccbcba", "cbbabacca", "aacabccb", "ccbb", "bcbacaa", "cbcca", "babca", "acccb", "acbbbca", "babaa", "bbcbca", "aaccab", "bab", "bbb", "b", "cbb", "bbbc", "abcc", "bbaba", "ac", "cab", "b", "baa", "accbb", "abaac", "ccc", "cca", "baac", "cbccb", "abbaa", "bbcaabc", "c", "ab", "ba", "cbc", "cbac", "bccba", "aaccbabc", "bbcbaacac", "ccbaa", "acca", "abccaa", "ac", "aaccbbab", "cba", "aaaccb", "aaabcb", "bacc", "acca", "ccbab", "cabaa", "bc", "cac", "acbccbaa", "cbacb", "bbcbac", "cbabccbaa", "bacca", "aba", "bcacabbca", "bccabc", "aacb", "abbcb", "acbcba", "cbac", "aacacb", "cac", "accbb", "cabcba", "a", "cbaaacb", "bbc", "cbabac", "babcbcc", "cbabcc", "cbaccab", "ccbbabaca", "abbbac", "bcca", "cabbba", "ca", "cbacbc", "aacbbacc", "cac", "bca", "caacbb", "bbacacc", "ccba", "abbcabc", "baba", "bacba", "bcbaacb", "acbbba", "aaa", "bbbacac", "acb", "cbcaacb", "cabcaac", "aabcbbcc", "bcbcacbaa", "baaccbc", "bcca", "abcb", "cac", "bcbac", "bcaabc", "babc", "aaccb", "baacba", "accabbcb", "cb", "ca", "ccbbaaab", "caccbbaab", "bbca", "bcacaabc", "cabbcb", "abca", "cabcba", "cc", "caabbbac", "abbaa", "caccb", "aa", "baca", "aacbbcb", "cbbaacba", "bac", "caabbbca", "bbabccca", "aabccab", "bcacbbc", "cbcaaa", "ccaab", "bcba", "cba", "b", "aabb", "abcaacb", "baccabba", "bcac", "bb", "acac", "bbacbcc", "cb", "accbb", "ccbbcbaaa", "babca", "aaccc", "cbbcac", "aabccab"], 150)	96
dostawa(["cbab", "ccaaca", "acb", "babbaccc", "abc", "cbabca", "ccacbb", "acabcb", "bacbaacc", "ccbbca", "bca", "ccbcaa", "ccabc", "cbbaabc", "bcaab", "caabab", "babac", "aca", "bacab", "aabacbcb", "acbc", "bac", "ccb", "bac", "caac", "ba", "ccaab", "ccba", "bac", "acbcab", "ba", "acbcac", "cbac", "aac", "abcbbc", "aacc", "aaccbca", "ccc", "abcabac", "acbbaca", "cbcca", "ca", "ac", "acccb", "aaabbcc", "aaa", "cabbac", "baab", "accbcb", "ccb", "bccaca", "cbcac", "ab", "cab", "bc", "cc", "b", "cbaacbc", "bba", "cbacbcbaa", "aa", "bcabaac", "caaaccb", "aca", "baca", "acbacb", "bacaa", "ccacbbaab", "abcba", "bcb", "abccb", "baaabc", "cabc", "cabccba", "bc", "cacbaacb", "abbcb", "cbca", "abcca", "ccacb", "bcac", "abbcaa", "cabcba", "baccb", "bbca", "abbcaa", "cabcbcab", "ac", "cbccbb", "ccb", "acabbbac", "cbcaa", "caabcbb", "babca", "bab", "cbaccbbaa", "cba", "cabbb", "ababacccb", "aabcbcba", "bcabcba", "bcbacab", "aabc", "cbaaacc", "cccab", "aba", "aaaccb", "bcbba", "cabbaba", "bb", "caa", "bab", "bacca", "bcbbc", "ccbbbc", "babc", "caacac", "cabaa", "bccab", "bcba", "bab", "ccbaac", "babba", "acacbb", "baabc", "bc", "cabbaac", "ca", "babac", "bcca", "caca", "abacca", "caabb", "cac", "bb", "acbb", "b", "bacbbcaa", "bcbc", "cbabbc", "cb", "bcbabaca", "ccabb", "bcbba", "cabacbba", "babba", "abccbb", "abcba", "baabcbca", "cc", "b", "aa", "cbbbca", "accbb", "accaabb", "acbccaa", "bbcacaa", "aabccab", "ccbca", "bacb", "cbacba", "aca", "ac", "cc", "aaacbcb", "ababccab", "bcb", "ccbac", "caca", "a", "bbcaba", "aa", "ccaacbb", "aacabb", "abcccaabb", "cbacaa", "babcabc", "bccaa", "aacb", "cbcabbaca", "baacc", "cbcca", "bcabaa", "abc", "cbcbaa", "cccabaab", "cacbcab", "abbaca", "abaac", "abacba", "abac", "baabc", "bcaaca", "bc", "cbbaa", "bba", "cbaacbbc", "accbbb", "aacbaccb", "abcabac"], 250)	156

Zadanie Trójki palindromów – LOGIA 19 SP (2018/19), etap 3

Treść zadania

Ania bada liczby. Ostatnio zaintrygowało ją, czy można daną liczbę zapisać w postaci sumy trzech różnych, dodatnich liczb palindromicznych, tj. takich, które czytane od początku i końca są identyczne. Pomóż Ani i napisz jednoparametrową funkcję pali, której parametrem jest liczba nie mniejsza niż 10 i nie większa niż 100 000. Wynikiem funkcji jest liczba różnych trójek liczb palindromicznych, których suma jest równa liczbie podanej jako parametr. Postaraj się tak rozwiązać zadanie, aby nie trzeba było długo czekać na wynik. Ocenie podlega też czas działania Twojego rozwiązania.

Przykłady:

Wynikiem **pali(100)** jest **7**. Trójki palindromów to: 1 11 88, 1 22 77, 1 33 66, 1 44 55, 3 9 88, 4 8 88, 5 7 88.

Wynikiem **pali(27)** jest **3**. Trójki palindromów to: 1 4 22, 2 3 22, 7 9 11.

Omówienie rozwiązania

Pomysł rozwiązania opiera się na realizacji algorytmu w dwóch krokach: najpierw generujemy uporządkowaną rosnąco listę wszystkich palindromów z rozpatrywanego przedziału, a potem sprawdzamy, ile sum trójek palindromów z tej listy jest równych parametrowi funkcji.

Sprawdzanie, czy dana liczba jest palindromem, można zaimplementować porównując czy napis utworzony z liczby jest równy napisowi będącego jego odwrotnością. Jeśli sprawdzamy trójki liczb palindromicznych warto zadbać o to, by nie sprawdzać tych samych trójek wielokrotnie. Dlatego iterujemy po pierwszej i drugiej liczbie z trójki, przy czym druga liczba jest zawsze większa niż pierwsza. Trzecią liczbę wyliczamy jako różnicę parametru funkcji i sumy liczb pierwszej oraz drugiej. Następnie sprawdzamy, czy znajduje się ona na wygenerowanej wcześniej liście liczb palindromicznych. Dodatkowo sprawdzamy, czy ta liczba jest większa od pozostałych, inaczej ta trójka była już rozpatrywana.

Lepsze pod względem złożoności czasowej jest rozwiązanie kombinatoryczne, ale opracowanie algorytmu jest bardziej skomplikowane, dlatego je pomijamy.

Rozwiązanie w języku Python

Funkcja `czy_pali(x)` sprawdza, czy danych napis jest palindromem. Zapis `napis[::-1]` wyodrębnia z napisu wszystkie znaki od początku do końca, ale w odwrotnej kolejności, czyli odwraca łańcuch znaków.

```
1.  def czy_pali(x):
2.      return str(x) == str(x)[::-1]
3.
4.  def pali(liczba):
5.      # tworzenie listy liczb palindromicznych
6.      pom = []
7.      for i in range(1, liczba+1):
8.          if czy_pali(i):
9.              pom.append(i)
10.
11.     # sprawdzanie trójek liczb
12.     ile = 0
13.     for i in range(len(pom)):
14.         for j in range(i+1, len(pom)):
15.             x = pom[i]
16.             y = pom[j]
17.             z = liczba - x - y
18.             if z > y and z in pom:
19.                 ile += 1
20.     return ile
```

Testy

Testy obejmują proste przypadki, które można przeliczyć ręcznie. Warto też zwrócić uwagę na przypadki brzegowe. W tym zadaniu pewną trudność stanowi opracowanie algorytmu o dobrej złożoności czasowej. Jeśli na przykład będziemy sprawdzać wszystkie możliwe trójki liczb, algorytm będzie działać dłużej. Dlatego warto uwzględnić w testach przypadki dla dużej wartości parametru.

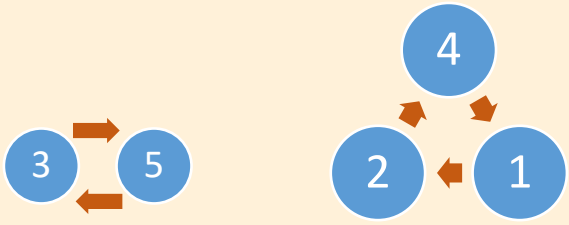
Wywołanie – Python	**Wynik**
pali(17)	9
pali(24)	4
pali(383)	23
pali(384)	33
pali(706)	117
pali(707)	65
pali(10000)	135
pali(10002)	134
pali(98998)	890
pali(99998)	7966

Zadanie System powiadamiania – LOGIA 19 SP (2018/19), etap 3

Treść zadania

W drużynie skautów został wprowadzony system powiadamiania polegający na tym, że po otrzymaniu informacji od drużynowego skaut jest zobowiązany przesłać informację do konkretnego skauta, a ten dalej do kolejnego itd., aż informacja dotrze z powrotem do skauta, który otrzymał informację od drużynowego. Każdy skaut ma przypisany numer i zna numer skauta, którego ma powiadomić. System powiadamiania opisany jest listą liczb. Numer elementu listy (pierwszy element listy ma numer 1) określa numer skauta, który wysyła informację. Wartość elementu listy określa numer skauta, do którego należy przesłać informację.

Na przykład lista złożona z liczb 2, 4, 5, 1, 3 oznacza, że skaut numer 1 powiadamia skauta numer 2, ten powiadamia skuta numer 4, skaut numer 3 powiadamia skauta numer 5, skaut numer 4 skauta numer 1, a skaut numer 5 skauta numer 3.



Napisz jednoparametrową funkcję **ile**, której parametrem jest niepusta lista opisująca system powiadamiania. Lista może mieć długość co najwyżej 1 000. Wynikiem jest minimalna liczba skautów, których musi powiadomić drużynowy, aby informacja dotarła do wszystkich skautów.

Przykłady:

Wynikiem **ile([2, 4, 5, 1, 3])** jest **2**. Wystarczy powiadomić skautów o numerach np. 3 i 4.

Wynikiem **ile([7,6,4,3,1,2,5])** jest **3**. Wystarczy powiadomić skautów o numerach np. 1, 2, 3.

Omówienie rozwiązania

Lista liczb opisująca system powiadamiania opisuje podział drużyny skautów na rozłączne grupy skautów – zbiory, w których wystarcza powiadomić jednego skauta, by informacja dotarła do wszystkich z grupy. Zadanie sprowadza się więc do znalezienia liczby cykli.

Będziemy potrzebować pomocniczej zmiennej `rozpatrzone`, która początkowo jest pusta. W kolejnych krokach będziemy dodawać elementy już rozpatrzone. Najpierw bierzemy pierwszy element listy, oznaczamy go `wsk` i dodajemy do listy rozpatrzonych elementów. Sprawdzamy, na który element wskazuje. Wskazywany element dodajemy do listy rozpatrywanych elementów i również badamy, na który element wskazuje. Postepowanie kontynuujemy, aż nie dojdziemy z powrotem do pierwszego elementu, czyli elementu `wsk`. Mamy wszystkie elementy cyklu, wobec czego zwiększamy licznik cykli.

Następnie bierzemy drugi element z listy, i o ile nie jest już w rozpatrywanych elementach, znajdujemy wszystkie elementy, na które wskazuje bezpośrednio i pośrednio. Takie badanie przeprowadzamy dla wszystkich elementów danej listy. Znaleziona liczba cykli jest odpowiedzią na zadanie.

OEiiZK
Ośrodek Edukacji Informatycznej
i Zastosowań Komputerów w Warszawie

Rozwiązanie w języku Python

Do przechowywania rozpatrzonych elementów można użyć listy lub zbioru. Gdy rozpatrujemy elementy listy, należy zwrócić uwagę, że elementy listy w Pythonie są numerowane od 0. Wobec tego element listy wskazuje na element – 1.

```
1.  def ile(lista):
2.      licznik = 0
3.      rozpatrzone = []
4.      for i in lista:
5.          if i not in rozpatrzone:
6.              #znajdowanie wszystkich elementów w cyklu
7.              rozpatrzone.append(i)
8.              wsk = lista[i-1]
9.              while wsk != i:
10.                 rozpatrzone.append(wsk)
11.                 wsk = lista[wsk-1]
12.             licznik = licznik + 1
13.     return licznik
```

Testy

Testy obejmują proste przypadki, która można bez trudu przeanalizować oraz trudniejsze, które obejmują różne przypadki testowe. Część z nic została wygenerowana losowo.

Wywołanie – Python	**Wynik**
ile([3, 1, 2])	1
ile([10, 8, 9, 7, 6, 5, 4, 3, 2, 1])	4
ile([2, 19, 12, 16, 14, 11, 17, 6, 20, 4, 7, 3, 10, 8, 13, 18, 5, 15, 1, 9])	5
ile([15, 13, 5, 14, 12, 3, 7, 10, 4, 2, 1, 6, 19, 8, 17, 16, 11, 18, 20, 9])	6
ile([3, 6, 8, 10, 2, 5, 7, 9, 4, 1])	3
ile([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])	10
ile([20, 71, 19, 3, 88, 97, 69, 10, 33, 22, 31, 56, 78, 47, 65, 18, 87, 77, 23, 72, 11, 25, 2, 79, 63, 43, 48, 74, 57, 94, 21, 52, 73, 62, 96, 39, 38, 81, 54, 14, 29, 59, 16, 42, 30, 86, 7, 4, 64, 100, 27, 50, 36, 90, 1, 55, 28, 95, 17, 40, 46, 15, 83, 70, 82, 75, 8, 91, 44, 24, 45, 58, 80, 67, 93, 98, 76, 5, 92, 85, 89, 34, 51, 61, 13, 84, 9, 32, 41, 53, 12, 26, 37, 66, 99, 68, 6, 49, 60, 35])	8
ile([70, 62, 21, 29, 87, 92, 63, 90, 2, 41, 50, 55, 86, 93, 38, 74, 33, 81, 94, 89, 3, 44, 68, 16, 51, 19, 76, 36, 46, 65, 91, 37, 17, 18, 85, 82, 54, 8, 61, 34, 73, 99, 96, 6, 9, 75, 32, 14, 11, 52, 26, 71, 15, 95, 40, 79, 1, 28, 30, 64, 58, 27, 98, 23, 24, 78, 5, 12, 35, 42, 20, 25, 66, 57, 100, 67, 88, 59, 10, 7, 77, 83, 72, 48, 4, 97, 84, 47, 13, 43, 39, 49, 60, 31, 45, 22, 80, 53, 56, 69])	7
ile([394, 25, 177, 893, 697, 43, 540, 528, 266, 404, 21, 898, 267, 745, 671, 663, 379, 304, 658, 67, 428, 731, 582, 669, 40, 285, 797, 12, 592, 138, 135, 764, 868, 875, 580, 502, 942, 469, 712, 906, 729, 102, 554, 576, 209, 497, 10, 208, 297, 279, 223, 776, 362, 532, 11, 859, 825, 41, 877, 624, 816, 159, 273, 613, 275, 870, 470, 467, 664, 815, 902, 274, 766, 47, 952, 835, 367, 854, 636, 277, 691, 306, 105, 396, 372, 872, 699, 19, 698, 983, 635, 587, 628, 943, 653, 572, 631, 354, 286, 386, 995, 26, 165, 538, 199, 852, 70, 905, 84, 637, 191, 553, 752, 760, 136, 759, 689, 119, 57, 332, 773, 951, 570, 74, 801, 389, 707, 955, 390, 703, 293, 110, 460, 152, 600, 122, 738, 260, 210, 768, 750, 677, 178, 987, 876, 515, 924, 565, 240, 887, 718, 189, 673, 114, 798, 753, 686, 556, 761, 651, 723, 567, 915, 947, 856, 779, 551, 649, 607, 894, 101, 369, 504, 145, 988, 966, 473, 605, 370, 606, 305, 583, 355, 660, 283, 863, 20, 95, 256, 103, 415, 343, 769, 720, 708, 120, 970, 581, 212, 991, 990, 477, 986, 527, 866, 450, 681, 180, 299, 328, 727, 62, 617, 229, 252, 422, 639, 46, 578, 307, 242, 573, 170, 52, 115, 999, 704, 480, 173, 923, 33, 632, 842, 732, 749, 93, 118, 303, 440, 543, 792, 365, 710, 914, 366, 771, 349, 883, 23, 865, 200, 541, 693, 684, 76, 483, 190, 272, 345, 187, 39, 220, 666, 398, 18, 429, 368, 294, 830, 416, 1000, 614, 819, 512, 568, 245, 247, 929, 498, 787, 45, 36, 186, 22, 828, 106, 13, 420, 564, 411, 844, 577, 316, 393, 148, 944, 489, 98, 648, 333, 694, 957, 843, 111, 945, 675, 203, 579, 982, 311, 465, 846, 7, 383, 144, 350, 376, 930, 268, 438, 149, 774, 222, 53, 913, 858, 901, 721, 48, 977, 213, 938, 519, 757, 546, 462, 612, 821, 667, 175, 857, 456, 491, 495, 861, 318, 838, 184, 790, 174, 728, 380, 8, 271, 701, 205, 363, 443, 716, 900, 327, 891, 881, 800, 300, 623, 264, 937, 280, 72, 337, 791, 323, 882, 211, 596, 82, 341, 251, 61, 545, 319, 611, 644, 975, 27, 939, 733, 439, 841, 448, 832, 112, 926, 889, 426, 451, 406, 954, 357, 557, 962, 89, 522, 936, 90, 321, 833, 109, 155, 227, 608, 87, 250, 571, 374, 804, 655, 633, 51, 949, 927, 284, 158, 254, 692, 772, 143, 685, 908, 630, 524, 793, 168, 961, 795, 569, 236, 963, 142, 137, 35, 315, 892, 874, 808, 444, 77, 851, 523, 326, 75, 356, 94, 289, 492, 826, 650,	16

181, 325, 558, 446, 176, 973, 322, 475, 827, 91, 503, 310, 131, 584, 978, 215, 688, 34, 739, 895, 410, 547, 185, 884, 334, 871, 552, 505, 425, 197, 130, 291, 146, 37, 461, 206, 320, 100, 466, 86, 71, 934, 741, 501, 510, 454, 928, 873, 850, 399, 127, 196, 9, 435, 981, 230, 997, 436, 85, 253, 432, 823, 574, 314, 799, 169, 809, 743, 452, 214, 358, 392, 183, 139, 589, 474, 591, 107, 670, 216, 147, 904, 969, 518, 550, 965, 676, 281, 661, 423, 68, 862, 156, 654, 734, 903, 740, 555, 441, 520, 730, 508, 994, 262, 382, 860, 585, 869, 361, 781, 806, 878, 290, 782, 919, 886, 989, 680, 535, 746, 601, 529, 134, 918, 54, 385, 255, 625, 140, 172, 126, 151, 270, 744, 63, 964, 933, 831, 588, 244, 49, 331, 378, 980, 717, 182, 767, 44, 201, 593, 756, 864, 312, 265, 499, 258, 803, 163, 129, 544, 419, 179, 715, 364, 912, 478, 276, 525, 530, 896, 726, 400, 295, 789, 397, 621, 104, 679, 421, 719, 234, 785, 971, 879, 976, 516, 3, 794, 678, 373, 81, 619, 537, 762, 829, 352, 561, 802, 836, 192, 514, 409, 259, 911, 839, 643, 820, 248, 559, 225, 594, 121, 979, 616, 261, 647, 330, 417, 445, 536, 59, 449, 748, 249, 985, 888, 29, 700, 713, 909, 388, 609, 233, 405, 221, 622, 329, 64, 347, 657, 28, 777, 360, 16, 468, 282, 542, 401, 867, 837, 615, 92, 457, 953, 807, 431, 812, 722, 348, 638, 424, 916, 548, 818, 232, 1, 202, 885, 778, 269, 463, 996, 302, 418, 511, 920, 73, 219, 485, 336, 408, 5, 880, 672, 824, 442, 811, 736, 706, 395, 427, 141, 822, 97, 472, 682, 157, 618, 38, 931, 513, 910, 430, 239, 500, 204, 765, 194, 124, 755, 786, 296, 629, 849, 263, 246, 763, 534, 166, 313, 935, 645, 83, 796, 751, 602, 890, 32, 123, 531, 308, 941, 80, 65, 160, 359, 709, 695, 496, 662, 735, 834, 526, 783, 590, 788, 353, 6, 238, 188, 959, 162, 948, 641, 974, 742, 243, 453, 853, 207, 193, 784, 412, 150, 562, 342, 683, 714, 754, 599, 24, 171, 998, 907, 488, 517, 237, 899, 60, 31, 476, 659, 984, 42, 967, 724, 257, 696, 458, 309, 925, 687, 775, 960, 17, 15, 218, 533, 634, 656, 344, 603, 917, 855, 549, 897, 381, 747, 241, 993, 922, 292, 58, 50, 459, 486, 167, 298, 132, 506, 493, 228, 154, 586, 338, 598, 958, 471, 403, 482, 69, 597, 164, 627, 4, 705, 317, 198, 278, 620, 339, 604, 507, 287, 521, 99, 226, 847, 509, 30, 391, 79, 224, 665, 813, 375, 481, 235, 840, 133, 652, 66, 640, 195, 128, 563, 377, 96, 690, 351, 335, 940, 301, 560, 2, 324, 972, 626, 88, 539, 956, 402, 932, 566, 817, 340, 125, 758, 161, 770, 610, 805, 437, 595, 217, 702, 487, 946, 113, 447, 479, 78, 117, 490, 464, 737, 668, 845, 810, 575, 814, 371, 992, 484, 56, 55, 725, 780, 674, 434, 384, 231, 494, 116, 950, 642, 288, 153, 346, 455, 921, 14, 407, 433, 646, 387, 968, 108, 414, 711, 413, 848])	
ile([181, 847, 101, 347, 945, 682, 587, 790, 511, 553, 932, 619, 734, 1000, 484, 854, 680, 593, 924, 470, 975, 332, 79, 440, 818, 875, 796, 711, 509, 596, 176, 152, 491, 383, 297, 384, 281, 969, 573, 574, 305, 962, 876, 518, 138, 81, 615, 40, 306, 404, 974, 510, 998, 963, 603, 688, 477, 322, 660, 299, 706, 300, 139, 873, 636, 645, 623, 382, 280, 570, 877, 722, 38, 557, 226, 274, 125, 666, 525, 334, 516, 19, 821, 402, 295, 867, 726, 756, 263, 24, 80, 667, 777, 995, 370, 637, 609, 98, 255, 148, 314, 647, 513, 530, 292, 438, 725, 430, 943, 253, 215, 186, 96, 250, 956, 399, 504, 744, 812, 462, 617, 11, 892, 712, 801, 288, 921, 638, 825, 542, 683, 654, 121, 179, 396, 902, 927, 94, 235, 648, 77, 656, 412, 310, 745, 330, 364, 12, 857, 376, 212, 439, 819, 381, 8, 47, 335, 241, 379, 990, 767, 506, 127, 147, 109, 146, 13, 264, 549, 353, 523, 764, 976, 32, 120, 112, 737, 715, 718, 361, 717, 900, 355, 177, 704, 377, 588, 930, 129, 514, 210, 572, 221, 611, 415, 650, 144, 222, 261, 810, 886, 7, 160, 651, 517, 243, 294, 881, 163, 592, 899, 806, 126, 320, 697, 643, 333, 122, 658, 141, 533, 111, 200, 986, 702, 486, 360, 548, 750, 401, 105, 989, 175, 338, 746, 88, 672, 489, 371, 420, 394, 912, 203, 237, 784, 824, 907, 977, 240, 201, 339, 270, 584, 622, 772, 390, 564, 476, 577, 578, 946, 827, 808, 137, 532, 861, 385, 856, 898, 988, 770, 809, 904, 188, 953, 536, 248, 39, 479, 78, 600, 28, 84, 894, 419, 997, 966, 613, 115, 948, 872, 136, 618, 855, 848, 951, 110, 362, 345, 123, 397, 866, 561, 498, 185, 657, 95, 190, 202, 426, 515, 69, 535, 887, 571, 133, 475, 795, 249, 944, 741, 257, 478, 829, 928, 897,	13

991, 85, 162, 50, 367, 699, 262, 389, 260, 60, 208, 714, 964, 845, 980, 859, 174, 244, 268, 598, 251, 17, 540, 65, 82, 747, 368, 73, 403, 673, 659, 716, 708, 503, 44, 407, 589, 888, 844, 490, 199, 652, 374, 369, 893, 563, 55, 813, 189, 732, 646, 366, 909, 392, 981, 994, 273, 343, 48, 869, 349, 473, 751, 918, 703, 117, 158, 911, 445, 895, 52, 16, 935, 671, 689, 400, 169, 312, 180, 817, 720, 130, 567, 605, 354, 481, 340, 541, 834, 143, 495, 959, 34, 165, 380, 931, 730, 195, 670, 616, 496, 762, 358, 290, 870, 925, 296, 954, 442, 569, 528, 560, 487, 153, 410, 826, 206, 709, 22, 843, 393, 731, 773, 811, 889, 551, 828, 815, 698, 860, 996, 789, 799, 937, 159, 794, 626, 768, 545, 71, 421, 552, 408, 5, 707, 719, 950, 474, 61, 279, 246, 469, 284, 565, 331, 205, 779, 214, 608, 906, 395, 526, 220, 814, 327, 309, 344, 879, 90, 724, 550, 766, 301, 164, 837, 942, 742, 317, 992, 579, 967, 987, 53, 769, 423, 582, 242, 663, 917, 18, 1, 128, 802, 328, 424, 388, 749, 852, 885, 691, 807, 662, 113, 480, 66, 458, 941, 665, 519, 792, 217, 497, 841, 482, 485, 468, 21, 350, 365, 507, 89, 275, 91, 624, 447, 232, 728, 585, 135, 465, 778, 172, 524, 639, 753, 607, 308, 556, 463, 391, 738, 755, 372, 630, 104, 805, 576, 209, 903, 302, 75, 729, 63, 114, 134, 409, 739, 743, 522, 406, 538, 27, 76, 842, 272, 454, 228, 625, 124, 155, 436, 831, 265, 455, 849, 411, 791, 759, 568, 884, 37, 686, 416, 64, 213, 49, 72, 106, 431, 67, 581, 151, 853, 43, 983, 797, 627, 171, 733, 359, 788, 838, 547, 816, 425, 471, 798, 286, 319, 736, 558, 939, 373, 161, 413, 103, 457, 252, 245, 289, 271, 256, 341, 167, 417, 534, 325, 375, 512, 661, 107, 307, 878, 606, 170, 880, 9, 575, 502, 446, 947, 116, 957, 453, 634, 970, 464, 701, 192, 679, 267, 713, 434, 184, 233, 765, 883, 916, 422, 460, 259, 723, 461, 668, 961, 748, 304, 387, 46, 346, 428, 787, 99, 984, 520, 62, 156, 183, 669, 336, 234, 690, 621, 283, 631, 33, 882, 644, 321, 386, 26, 566, 357, 915, 505, 437, 316, 429, 20, 342, 527, 664, 337, 710, 910, 590, 864, 229, 908, 820, 313, 494, 223, 493, 443, 919, 348, 449, 131, 591, 685, 29, 500, 555, 653, 157, 913, 923, 839, 758, 786, 580, 15, 965, 979, 501, 145, 293, 940, 168, 780, 783, 198, 521, 858, 182, 351, 785, 677, 326, 142, 874, 450, 531, 968, 191, 207, 601, 239, 850, 225, 687, 763, 761, 695, 93, 727, 456, 318, 311, 633, 774, 247, 640, 108, 427, 324, 832, 863, 451, 823, 840, 871, 793, 926, 291, 287, 193, 890, 629, 922, 227, 800, 154, 219, 197, 952, 597, 2, 754, 539, 414, 435, 972, 42, 676, 488, 896, 649, 868, 140, 933, 612, 4, 459, 3, 599, 620, 602, 258, 149, 985, 544, 74, 891, 562, 846, 204, 472, 329, 632, 862, 905, 224, 10, 781, 230, 398, 543, 554, 901, 323, 56, 194, 614, 934, 276, 30, 150, 958, 721, 266, 178, 35, 6, 58, 752, 83, 231, 92, 45, 628, 508, 277, 70, 835, 54, 441, 352, 604, 68, 978, 865, 87, 166, 822, 432, 700, 118, 949, 537, 675, 86, 14, 973, 757, 418, 282, 586, 57, 444, 378, 356, 655, 735, 23, 173, 102, 692, 635, 920, 119, 529, 914, 803, 238, 218, 938, 559, 595, 993, 236, 363, 100, 59, 693, 694, 216, 187, 775, 36, 492, 483, 466, 448, 678, 278, 41, 960, 642, 211, 285, 696, 776, 833, 31, 405, 782, 254, 546, 132, 771, 929, 936, 851, 51, 955, 641, 499, 315, 594, 705, 836, 740, 804, 830, 674, 681, 971, 982, 433, 684, 303, 196, 610, 269, 298, 760, 452, 583, 467, 97, 25, 999])	

Zadanie Urodziny –LOGIA 19 SP (2018/19), etap 3

Treść zadania

Bartek co roku obchodzi hucznie swoje urodziny. W tym roku postanowił, że będzie także celebrował swoje urodziny w te dni, które są odległe od jego dnia urodzin o wielokrotność 1000 dni. Niestety wyznaczenie dat, w których obchodziłby kolejne „tysiącznice" jest poza jego możliwościami. Szczególnie, że musi uwzględnić lata przestępne tj. takie, które są podzielne przez 4 i nie są podzielne przez 100 lub są podzielne przez 400. Bartek zapisuje daty w postaci słowa *uddmmrrrr*, gdzie *u* jest stałe, *dd* to dwucyfrowy numer dnia, *mm* – dwucyfrowy numer miesiąca, *rrrr* – czterocyfrowy numer roku (np. u13022004 oznacza datę 13 lutego 2004 roku).

Pomóż Bartkowi i napisz dwuparametrową funkcję **urodziny**, której pierwszym parametrem jest słowo określające datę urodzin Bartka, a drugim liczba nie mniejsza niż 1 i nie większa niż 100 określająca, którą „tysiącznicę" Bartek chce wyznaczyć. Wynikiem jest słowo określające datę, kiedy wypada kolejna „tysiącznica" określona drugim parametrem. Przyjmij, że pierwszy parametr i wynik tworzą poprawną datę z zakresu 01.01.1901 – 31.12.2499.

Przykłady:

wynikiem **urodziny('u13022004', 2)** jest **'u05082009'**, .

wynikiem **urodziny('u15011905', 10)** jest **'u02061932'**.

Omówienie rozwiązania

Zadanie Urodziny porusza często spotykany podczas programowania problem obliczania dat. Rozwiązanie wykorzystujące bibliotekę `datetime` jest proste, ale podczas konkursu nie można z niej korzystać (dozwolone są tylko biblioteki `turtle` i `math`). Z problemem wyliczania dat należy się zmierzyć bez pomocy funkcji bibliotecznych.

Odnotujmy podstawowe fakty o kalendarzu:

- liczba dni w kolejnych miesiącach roku jest różna,
- miesiąc luty dodatkowo może mieć 29 dni. Dzieje się tak w latach, które są podzielne przez 4, ale niepodzielne przez 100 – chyba, że są podzielne przez 400 – czyli lata 2100, 2200, 2300 nie będą przestępnymi.

Zadanie to rozwiążemy pisząc funkcję, która zlicza liczbę dni od pewnej ustalonej daty (np. 1 stycznia roku zerowego, której to daty w rzeczywistości nie było) oraz drugą funkcję, która jest funkcją odwrotną do pierwszej, czyli dla danej liczby dni zwraca datę.

W pseudokodzie pierwsza funkcja mogłaby wyglądać tak:

```
ndnia ← 365*rok + rok div 4 – rok div 100 + rok div 400 + md[m-1]+d
jeśli przestępny(rok) i m < 3
    ndnia ← ndnia – 1
wynik ndnia
```

W pierwszej linii obliczamy liczbę dni od wspomnianej daty 1 stycznia roku zerowego z uwzględnieniem lat przestępnych. Posługujemy się tablicą `md`, której kolejne elementy wynoszą [0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365] – jest to liczba dni, jakie upłynęły w poprzednich miesiącach danego roku i tutaj nie liczymy lat przestępnych. Zakładamy, że luty ma 28 dni. W drugiej linii korygujemy `ndnia`, gdy mamy rok przestępny, lecz nie było jeszcze 1 marca.

Funkcja **przestępny(rok)** mogłaby wyglądać tak:

```
zwróć wartość wyrażenia logicznego:
(rok mod 4 = 0) oraz ((rok mod 100 <> 0) lub (rok mod 400 = 0))
```

Wynikiem jest PRAWDA, gdy wartość `rok` odpowiada rokowi przestępnemu. Operatory `div` i `mod`, to odpowiednio dzielenie całkowitoliczbowe i reszta z dzielenia.

Funkcja realizująca obliczenia w drugą stronę, tzn. z danego numeru dnia wyliczająca datę podzielona została na dwie części. W części pierwszej jest wyznaczany rok, w którym data się znajduje – są to kroki 1-4 z listingu poniżej. W drugiej części wyliczany jest miesiąc i dzień tego roku.

```
rok  ← ndnia // 365
dopóki ndnia – (365 * rok + rok // 4 - rok // 100 + rok // 400) <= 0
    rok ← rok – 1
ndnia ← ndnia - (365 * rok + rok div 4 - rok div 100 + rok div 400)
jeśli ndnia > 365
    ndnia ← ndnia – 366
    rok ← rok + 1
jeśli przestępny(rok)
    jeśli ndnia = 31 + 28
          wynik 29, 2, rok
    jeśli ndnia < 31 + 29
          ndnia ← ndnia + 1
miesiąc ← 1
dopóki ndnia – md[miesiąc] > 0
    miesiąc ← miesiąc + 1
ndnia ← ndnia – md[miesiąc – 1]
wynik ndnia, miesiąc, rok
```

Gdyby każdy rok zawierał 365 dni, to obliczenie roku zakończylibyśmy wyrażeniem:

```
rok = ndnia // 365
```

Z uwagi, że część dni z puli o wartości `ndnia` znajduje się w latach przestępnych należy skorygować tak obliczoną wartość zmiennej `rok`. Zrobione to zostało w pętli z kroku 2. Pętla dla podanych zakresów lat wykona się dwa razy – gdyż liczba lat przestępnych w latach 1-1901 i 1-2499 są odpowiednio równe 460 i 606.

Krok 4 zawiera korektę, gdy data wypada 1 stycznia roku przestępnego. Dzień ten już był odejmowany w kroku 3, a w rzeczywistości nie powinniśmy go odejmować. Podobna korekta jest w kroku 5, lecz tu nie musimy już zmieniać roku – wystarczy odjąć jeden od `ndnia`.

W krokach 6-7 wyznaczamy numer miesiąca wykorzystując wcześniej omawianą tablicę `md`. W końcu po odjęciu od `ndnia` liczby dni, jaka upłynęła od początku roku do końca poprzedniego miesiąca, otrzymujemy numer dnia w miesiącu.

Mając te dwie funkcje obliczenie „tysiącznicy" sprowadza się do wywołania:

```
ndnia2data( data2ndnia(data) + n * 1000 )
```

Pozostaje jeszcze sformatowanie zwróconej daty zgodnie ze specyfikacją.

Rozwiązanie w języku Python

```
1.  def data2ndnia(data):
2.    d = int(data[1:3])
3.    m = int(data[3:5])
4.    r = int(data[5:])
5.    ndnia = 365 * r  + md[m - 1] + d + r // 4 - r // 100 + r // 400
6.    if r % 4 == 0 and (r % 100 != 0 or r % 400 == 0) and m < 3: ndnia -= 1
7.    return ndnia
```

W liniach 2-4 pobieramy z łańcucha tekstowego z datą odpowiednie fragmenty z dniem miesiąca, miesiącem i rokiem. W linii 5 obliczamy numer dnia i ewentualnie korygujemy, gdy mamy rok przestępny i data jest przed 1 marca.

Wypełnienie tablicy `md` zrealizowano za pomocą kodu:

```
8.  md = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 ]
9.  for i in range(1, len(md)):
10.   md[i] = md[i - 1] + md[i]
```

Realizacja funkcji odwrotnej, czyli z numeru dnia wyznaczającej datę wygląda następująco:

```
11. def ndnia2data(ndnia):
12.   r = ndnia // 365
13.   while ndnia - (365 * r + r // 4 - r // 100 + r // 400) <= 0:
14.     r -= 1
15.   ndnia = ndnia - (365 * r + r // 4 - r // 100 + r // 400)
16.   if ndnia > 365:
17.     ndnia -= 366
18.     r += 1
19.   if r % 4 == 0 and (r % 100 != 0 or r % 400 == 0):
20.     if ndnia == 31 + 28: return 29, 2, r
21.     if ndnia < 31 + 29: ndnia += 1
22.   m = 1
23.   while ndnia - md[m] > 0: m += 1
24.   ndnia = ndnia - md[m - 1]
25.   return ndnia, m , r
```

Zwracana jest trzyelementowa krotka z dniem miesiąca, miesiącem i rokiem.

Obie wyżej omówione funkcje oraz tablicę `md` umieszczono wewnątrz funkcji **urodziny**. Z krotki, którą zwraca funkcja `ndnia2data`, jest w linii 27 formatowany łańcuch z pierwszym znakiem 'u', a następnie dwie dwuznakowe i jeden czteroznakowy z dopełnianiem zerami wiodącymi np. 01082003. Operator * w funkcji `format` służy do rozpakowania krotki na trzy argumenty.

```python
def urodziny(data, n):
    md = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 ]
    for i in range(1, len(md)):
        md[i] = md[i - 1] + md[i]
    def data2ndnia(data):
        d = int(data[1:3])
        m = int(data[3:5])
        r = int(data[5:])
        ndnia = 365 * r  + md[m - 1] + d + r // 4 - r // 100 + r // 400
        if r % 4 == 0 and (r % 100 != 0 or r % 400 == 0) and m < 3: ndnia -= 1
        return ndnia
    def ndnia2data(ndnia):
        r = ndnia // 365
        while ndnia - (365 * r + r // 4 - r // 100 + r // 400) <= 0:
            r -= 1
        ndnia = ndnia - (365 * r + r // 4 - r // 100 + r // 400)
        if ndnia > 365:
            ndnia -= 366
            r += 1
        if r % 4 == 0 and (r % 100 != 0 or r % 400 == 0):
            if ndnia == 31 + 28: return 29, 2, r
            if ndnia < 31 + 29: ndnia += 1
        m = 1
        while ndnia - md[m] > 0: m += 1
        ndnia = ndnia - md[m - 1]
        return ndnia, m , r
    wy = ndnia2data( n * 1000 + data2ndnia(data))
    return 'u{:02}{:02}{:04}'.format(*wy)
```

Testy

Testy zostały tak dobrane, by sprawdzić obliczenia bez lat przestępnych, z latami przestępnymi oraz zawierały lata podzielne przez 100. Były też testy, w których poszukiwane daty wypadały 29 lutego lub 1 stycznia.

Wywołanie – Python	**Wynik**
urodziny("u06042093", 1)	"u01012096"
urodziny("u05041989", 1)	"u31121991"
urodziny("u16072377", 10)	"u01122404"
urodziny("u28071987", 6)	"u31122003"
urodziny("u03062337", 1)	"u28022340"
urodziny("u30122380", 7)	"u29022400"
urodziny("u13121901", 3)	"u01031910"
urodziny("u30122000", 7)	"u29022020"
urodziny("u02081991", 94)	"u12122248"
urodziny("u20111976", 11)	"u02012007"