Zadanie Strony – LOGIA 18 (2017/18), etap 3

Treść zadania

Marek oznaczył odwiedzane przez siebie strony internetowe kolejnymi małymi literami alfabetu łacińskiego. Każdą z nich odwiedził dokładnie raz, nie pamięta jednak dokładnie, w jakiej kolejności. Dla niektórych stron pamięta, którą odwiedził wcześniej, a którą później (np. że stronę *a* odwiedził przed *c*, a stronę *d* przed *b*).

Napisz dwuparametrową funkcję **strony**. Pierwszy parametr określa liczbę odwiedzonych stron i może przyjmować wartości od 2 do 8. Drugi jest listą (o długości nie większej niż 30) dwuliterowych słów określających kolejność odwiedzania wybranych stron (np. słowo *ab* oznacza, że strona *a* była odwiedzona wcześniej niż *b*). Wartością funkcji jest liczba możliwych kolejności odwiedzin wszystkich stron.

Przykłady:

wynikiem **strony(2,["ba"])** jest **1** (możliwa kolejność odwiedzania: *ba*),

wynikiem **strony(4,["ab","ac","ad","db"])** jest **3** (możliwe kolejności odwiedzania: *acdb*, *adbc*, *adcb*).

Omówienie rozwiązania

W zadaniu mamy dwa parametry – pierwszy określa liczbę symboli, które opisują odwiedzane strony, a drugi wskazuje na kolejność odwiedzanych stron. Zadanie sprowadza się do policzenia liczby permutacji spełniających warunki opisane drugim parametrem. Permutacja to ustawienie w dowolnej kolejności danych elementów. W zadaniu tymi elementami jest kilka początkowych małych liter alfabetu łacińskiego (ich liczbę określa pierwszy parametr). Najpierw zajmiemy się algorytmem generowania permutacji. Różnych permutacji n elementów jest $n!$, dlatego ograniczony jest zakres danych w zadaniu (dla największej wartości parametru 8!=40320). Wprowadźmy następujące przekształcenie jednej permutacji w drugą: `rotacja(`$a_0a_1\ldots a_{n-1}$`,k)` oznacza ustawienie elementu a_k na pierwszej pozycji i przesunięciu elementów $a_0a_1\ldots a_{k-1}$ o jedną pozycję w prawo.

`rotacja(`$a_0a_1\ldots a_{n-1}$`,k)` = $a_ka_0a_1\ldots a_{k-1}a_{k+1}\ldots a_{n-1}$

Rozpatrzmy na przykładzie generowanie wszystkich permutacji trzech liter z wykorzystaniem rotacji. Rozpoczynamy od słowa `abc`. Zakładamy, że litera `a` (w tym momencie pierwszy element) ma indeks 0. Rozpoczynamy od największej wartości indeksu parametru przestawianego i kontynuujemy aż nie dojdziemy do permutacji, która już była.

1. abc
2. rotacja(abc,2) = cab
3. rotacja(cab,2) = bca
4. rotacja(bca,2) = abc — ta permutacja już była, więc jej nie uwzględniamy, zmniejszamy k
5. rotacja(abc,1) = bac
6. rotacja(bac,2) = cba — rozpoczynamy ponownie od największej wartości k
7. rotacja(cba,2) = acb
8. rotacja(acb,2) = bac — ta permutacja już była, więc jej nie uwzględniamy, zmniejszamy k
9. rotacja(bac,1) = abc — ta permutacja już była, więc jej nie uwzględniamy, zmniejszamy k

Rozpoznać sytuację, że dana permutacja już wystąpiła można po tym, że litera na *k*-tej pozycji wróciła na swoją pozycję wyjściową. Algorytm generowania permutacji kończy się, gdy *k* osiągnie wartość 0.

Zapis tego algorytmu jest następujący:

```
permutacja = a0a1…an-1
sprawdź permutację, czy spełnia warunki zadania
k = n-1
dopóki k>0 wykonuj
    permutacja = rotacja(permutacja,k)
    jeśli permutacja[k] jest różna od k-tej litery alfabetu
          sprawdź permutację, czy spełnia warunki zadania
          k = n-1
    w przeciwnym przypadku
          k = k-1
```

Sprawdzenie, czy dana permutacja spełnia warunki zadania polega na przejrzeniu wszystkich warunków i analizie, czy pierwsza litera warunku jest przed drugą w permutacji. W przypadku spotkania pierwszej sprzeczności permutację odrzucamy. Jeśli wszystkie warunki są spełnione, należy licznik „dobrych" permutacji powiększyć o jeden.

Innym możliwym sposobem rozwiązania jest rekurencyjne generowanie permutacji. Należy wówczas pamiętać, że licznik „dobrych" permutacji musi być deklarowany jako zmienna globalna poza funkcją rekurencyjną.

Rozwiązanie w języku Python

Ze względu na łatwość operowania permutację liter będziemy pamiętać na liście (elementami będą litery), a nie jako słowo. Warto zauważyć, że słowa w języku Python nie są modyfikowane. Generując początkową permutację wykorzystujemy listę składaną (linia kodu 3).

```
1.  def strony(n,warunki):
2.      ile = 0
3.      permutacja = [chr(i+97) for i in range(n)]
4.      if perm_ok(permutacja,warunki): ile += 1
5.      k = n-1
6.      while k>0:
7.          permutacja = rotacja(permutacja,k)
8.          if chr(k+97)!=permutacja[k]:
9.              if perm_ok(permutacja,warunki): ile += 1
10.             k = n-1
11.         else:
12.             k -= 1
13.     return ile
```

Kolejne małe litery alfabetu łacińskiego mają kody ASCII poczynając od 97, więc taką literę można uzyskać przy pomocy funkcji `chr(i+97)`, gdzie i zmienia się od 0 do wartości pierwszego parametru pomniejszonego o 1.

```
14. def perm_ok(permutacja,warunki):
15.     for war in warunki:
16.         p = permutacja.index(war[0])
17.         k = permutacja.index(war[1])
18.         if p>k:
19.             return False
20.     return True
```

Zmienne *p* i *k* (wiersze 3 i 4) w pętli przeglądającej wszystkie warunki przyjmują odpowiednio wartość indeksu pierwszej i drugiej litery warunku w permutacji. W przypadku znalezienia nieprawidłowej kolejności wartością funkcji jest fałsz. Operację *rotacja* można zapisać w Pythonie dzięki możliwości pobrania określonej liczby elementów z przodu lub tyłu listy. Zajmuje jedną linię kodu.

```
21. def rotacja(permutacja,k):
22.     return [permutacja[k]]+permutacja[:-(len(permutacja)-k)]+permutacja[k+1:]
```

Testy

Testowanie rozwiązania warto rozpocząć od przykładów, których wyniki można sprawdzić ręcznie np. podobnych do podanych w treści zadania. Potem można testować działanie programu dla innych danych, choć i w tym przypadku warto tak dobrać dane, aby można było ocenić poprawność wyniku. W testach powinny być też ujęte przypadki brzegowe, a więc gdy warunki spełniają wszystkie permutacje (lista warunków jest pusta), żadna (lista warunków jest sprzeczna) lub tylko jedna permutacja.

Wywołanie – Python	**Wynik**
strony(5, ["ba", "eb", "ea"])	20
strony(7, ["cd", "ce", "de", "af", "bc"])	105
strony(8, ["ab", "ac", "ad", "ae", "af", "ag", "ah", "hb", "hc", "hd", "he", "hf", "hg", "bc", "bd", "be", "bf", "bg", "gc", "gd", "ge", "gf", "cd", "ce", "cf"])	6
strony(6, [])	720
strony(3, ["ab", "ba"])	0
strony(3,["ab", "ac", "bc"])	1

Ponieważ pierwszy parametr przyjmuje wartości od 2 do 8, można sprawdzić działanie dla wszystkich wartości parametru.