Zadanie Ukryte wyrazy – LOGIA 22 (2021/22), etap 3

Treść zadania

Karol bawi się z Tomkiem w ukrywanie wyrazów w kwadratowej tabeli złożonej z 10 wierszy i 10 kolumn. Wybiera pole, w którym wpisze pierwszą literę ukrywanego wyrazu. Drugą literę wpisuje na sąsiednim polu (powyżej, poniżej, w prawo lub lewo). Z kolejnymi literami postępuje w ten sam sposób. Wpisując wyraz, nie wykorzystuje dwa razy tego samego pola. W pozostałe pola wpisuje losowe litery.

Zadaniem Tomka jest odnalezienie ukrytego wyrazu w tabeli. Tabela jest podana w postaci napisu złożonego ze 100 liter, pierwsze 10 liter określa pierwszy wiersz tabeli, kolejne 10 liter drugi wiersz, itd.

A	B	C	D	E	R	A	S	S	Q
M	G	Y	U	K	A	E	S	S	Q
T	R	W	O	I	S	O	A	B	C
U	I	H	K	A	I	W	Y	U	K
Y	E	W	A	G	H	O	S	S	Q
R	T	T	K	R	A	K	Y	U	K
W	Y	U	K	A	Y	U	K	S	Q
I	S	S	Q	A	S	S	Q	U	K
E	Y	Y	U	K	Y	U	K	S	Q
O	S	A	S	Q	A	Q	Y	U	K

W tabeli Karol ukrył wyraz „KRAKOWIAK”.

Pomóż Tomkowi i napisz program, który wczyta tabelę oraz ukryty wyraz i wypisze numer pierwszej litery wyrazu lub -1, gdy wyrazu nie można odnaleźć. Gdy wyraz jest ukryty kilkukrotnie, to wypisze najmniejszy numer.

Wejście:

Pierwszy wiersz: napis złożony ze 100 wielkich liter alfabetu łacińskiego, pierwsze 10 liter określa pierwszy wiersz tabeli, kolejne 10 liter drugi wiersz, itd.

Drugi wiersz: ukryty wyraz (o długości od 1 do 10) złożony z wielkich liter alfabetu łacińskiego.

Wyjście:

Liczba naturalna z przedziału [1;100] – najmniejszy numer pierwszej litery znalezionego wyrazu w tabeli lub -1, gdy wyraz nie występuje.

Przykład 1	
Wejście	**ABCDERASSQMGYUKAESSQTRWOISOABCUIHKAIWYUKYEWAGHOSSQRTTKRAKY UKWYUKAYUKSQISSQASSQUKEYYUKYUKSQOSASQAQYUK KRAKOWIAK**
Wyjście	**54**

Przykład 2	
Wejście	**AXCDERASSQMGLECAESSQTRWOISOAKCUWHKAIWYUKYEWAGHOSSRRTTKRAKMUKWTUKAYUKSQISSCASSQUWEUYEKYUKSQOSALQAQYUK** **CEL**
Wyjście	**15**
Przykład 3	
Wejście	**ABCDEFGHIJMABCDEFGHIIJABCDEFGHHIJABCDEFGGHIJABCDEFFGHIJABCDEEFGHIJABCDDEFGHIJKBCCDEFGHIAABBCDEFGHIJA** **NIEMA**
Wyjście	**-1**

Omówienie rozwiązania

Zaczynamy od wczytania danych – dwóch napisów. Pierwszy to plansza, drugi – to wyraz, którego będziemy szukać. Warto zauważyć, że plansza jest stałego rozmiaru 100 – 10 wierszy i 10 kolumn. Ułatwia to znajdowanie sąsiadów – pod warunkiem, że nie znajdujemy się na brzegu: lewy sąsiad – to pole o numerze o 1 mniejszym, prawy- o 1 większym, górny – o 10 mniejszym, a dolny o 10 większym.

Będziemy przeglądać kolejne pola planszy i sprawdzać, czy od tego pola można znaleźć ukryty wyraz. Jeśli tak, to program powinien wypisać numer tego pola, jeśli nie to przejść do następnego pola. Gdy zostaną przejrzane wszystkie pola, a wyraz nie zostanie znaleziony, to należy wypisać -1.

Szukanie realizujemy w funkcji rekurencyjnej, która dla danego numeru pola będzie sprawdzać, czy z niego rozpoczyna się ukryty wyraz. Będziemy wykorzystywać listę numerów pól odwiedzonych, by uniknąć sytuacji, w której bierzemy jedno pole dwa razy. Przy pierwszym wywołaniu funkcji inicjalizujemy tę listę jako listę pustą.

W samej funkcji na początku sprawdzamy, czy długość wyrazu nie jest 0, jeśli tak to znaleźliśmy wyraz, kończymy działanie funkcji – wynikiem jest prawda (`True`). Następnie sprawdzamy czy rozpatrywane pole jest różne od pierwszej litery wyrazu lub pole jest odwiedzone, wtedy też kończymy, ale wynikiem funkcji jest fałsz (`False`) – nie znaleźliśmy wyrazu. Potem dodajemy numer bieżącego pola do listy odwiedzonych i rekurencyjnie szukamy wyrazu dla pól sąsiednich. Przy czym w wywołaniu rekurencyjnym pomijamy bieżącą literę wyrazu. Jeśli wyraz zostanie znaleziony, kończymy działanie funkcji i przekazujemy wynik True.

Rozwiązanie w języku Python

W rozwiązaniu zaimplementowano następujące funkcje:

- `analizuj(plansza, wyraz)` – funkcja jest odpowiedzialna za przejrzenie kolejnych pól planszy i sprawdzanie czy od danego pola nie zaczyna się wyraz.
- `szukaj(plansza, nr, odwiedzone, wyraz)` – funkcja sprawdza, czy od pola o numerze nr zaczyna się wyraz. Jest to funkcja rekurencyjna, a wynikiem jest `True` – znaleziono wyraz lub `False` – brak wyrazu.

- `sasiedzi(nr)` – wynikiem funkcji jest lista numerów pól sąsiednich, przy czym sprawdzane jest czy dane pole nie jest na brzegu.

```
 1 def sasiedzi(nr):
 2     pom = []
 3     if nr%10 != 0:
 4         pom.append(nr - 1)
 5     if nr%10 != 9:
 6         pom.append(nr + 1)
 7     if nr >= 10 :
 8         pom.append(nr - 10)
 9     if nr < 90:
10         pom.append(nr + 10)
11     return pom
12
13 def szukaj(plansza, nr, odwiedzone, wyraz):
14     if len(wyraz) == 0:
15         return True
16     if (plansza[nr] != wyraz[0]) or (nr in odwiedzone):
17         return False
18     else:
19         odwiedzone.append(nr)
20         for sas in sasiedzi(nr):
21             if szukaj(plansza, sas, odwiedzone, wyraz[1:]):
22                 return True
23         odwiedzone.remove(nr)
24         return False
25
26 def analizuj(plansza, wyraz):
27     for nr in range(len(plansza)):
28         if szukaj(plansza, nr, [], wyraz):
29                 return nr + 1
30     return -1
31
32 plansza = input()
33 wyraz = input()
34 print(analizuj(plansza, wyraz))
```

Na etapie debugowania warto jeszcze dodać funkcję wypisującą planszę, nie jest ona jednak używana w gotowym rozwiązaniu.

```
1 def wypisz(plansza):
2     for i in range(10):
3         print(plansza[i*10:i*10+10])
```

Testy

Testy powinny obejmować różne długości i położenia wyrazu tak, aby przetestować poprawne znajdowanie sąsiadów w lewo, w prawo, w górę i w dół. Nie należy zapomnieć o przypadkach, gdy wyraz jest na brzegu, gdyż tam mogą pojawić się błędy.

Poniżej znajdują się przykładowe testy: pierwsze dwie linijki to dane, trzecia – wynik.

Test

ABCDERASSQMGYUKAESSQTRWOISOABCUIHKAIWYUKYEWAGHOSSQRTTKRAKYUKWYUKAYUKSQISS
QASSQUKEYYUKYUKSQOSASQAQYUK

KRAKOWIAK

54

Test

AXCDERASSQMGLECAESSQTRWOISOAKCUWHKAIWYUKYEWAGHOSSRRTTKRAKMUKWTUKAYUKSQIS
SCASSQUWEUYEKYUKSQOSALQAQYUK

CEL

15

Test

ABCDEFGHIJMABCDEFGHIIJABCDEFGHHIJABCDEFGGHIJABCDEFFGHIJABCDEEFGHIJABCDDEFGHIJKBC
CDEFGHIAABBCDEFGHIJA

NIEMA

-1

Test

TUIIPCGCKUUKOVYUHFQXQNYXNMYSPHYZJGEXVJTICQVCLCVAIKJRUTVMVYSPXRVMXDWFAZJYYWH
QQYXVACTMNTGWGZVFEJXJRNRA

D

66

Test

TVFEDLZDLQQLNNJCLHQPCMEKFETGPJUVPYTDRHGMVFMMXOAXFSBQSTIJSDRWUKORGRJWQJUJVB
MDXJHVKKVTAGNADYVVSQQFFVQD

NA

87

Test

LGNOJCYZCUXYZAHCTTPJOXHTZIPTOAJKCNYWNQAXJRNFLCZTRNEFGXARRWYYPAPFNALHHLNUGVVP
TNJCASQCJVQZGHAFPCCOVOOC

RAP

56

Test

BAIVUWMYNZTJQQAWXMCPBODLAUJBPDZECSWPQSLSKFFOMHFXYBFNYKOUMGJIXXAOTLXPWJDPJCRYMBOGYHYBTZTCVHMZXIZDKCHG

KOT

54

Test

JTDYCCSPODNQXUIOFDKDYSLDQYMHYBAEMEPINVTVBESGQQGARUYSVFDEOTDZOUCVMYTUHPEJTPIIDVEGLNOEEPTEBVQBAWSPTNAO

PIES

74

Test

KGTEHUDKYIIYIDWIREXKBEHTGAJLLURWVVFCTCRIHSNJWSZEAOPDWHSBLBMDQYUUDUCRASLVTDEQPJWSPGIYYCIXWCUFVESCBENR

ZEBRA

47

Test

AGXBRTDXNHWIJNMPWBOFRSSVUQNAWVHEPNUUMVMXQYMCVQAVSLZVFUPJSZSVNRABMRHMNRDZOKNRQNDHQOKOBGUCHLHROOWPEBEA

OKOKO

73

Test

EBNHQYXOQOBYZJEVWCZDKCVFMEWTBJTJVUIVOGBPTUEUZUFYZOPPJKPTOONCAPISUEITFFXUPEQMYOZWBPICZHUHBBXVRUGUECAA

ABBAC

100

Test

SOFAPPGQFPMWTNTUUQNQODKPVUEAABORFMZNIZHLRPTXKPWRSPMFQAJVAJACVTUZAWPWUYSRWBPLWADXJCEDFVPKZUFDQKCFRVNE

PAJAC

67

Test

BABBBCBAAABCABABACCCBCCACCACCACAACACCCBBAAABABACBBACBABBAAABABCCBCBBBCCCBBCBBCCCABBCABAABBCABBCCBBAA

BACBCBCBBC

44

Test

KDKDWDODDOOWWWDKDDDOKOKOODDWOWOWDODWWKOWDOWDWDKKDOOKWOKKKKWKK
OOOWKDWOKKWKWWOKKDOKODDDKKODOOWDOKKWKWW

DOKWODKOD

49

Test

ZYZZYZZXZXYXZZXYXXYXYYYXYYYZZYXYXYYXYYZYYZXZYZXXYZXXYXZXYYYXYZXYYXXZXYZXXZXZXYXZYYY
XXXYYXYYYZZYYZYYY

XYZXYZ

17