Zadanie 1 - Największa różnica w podciągu

Szablon rozwiązania: `zad1k.py`

Dany jest ciąg binarny tj. zer oraz jedynek S. Proszę znaleźć taki SPÓJNY fragment tego ciągu, w którym różnica pomiędzy ilością zer, a jedynek, będzie jak największa. Jeżeli w ciągu występują same jedynki, należy założyć, że rozwiązaniem jest -1

Algorytm należy zaimplementować jako funkcję postaci:

```
def roznica( S ):
  …
```

która przyjmuje ciąg S i zwraca wyliczoną największą osiągalną różnicę.

Przykład. Dla ciągu:

```
11000010001
```

Wynikiem jest liczba **6**

Testowanie Rozwiązań

Żeby przetestować rozwiązania należy wykonać polecenie: `python3 zad1k.py`

Zadanie 2 - Najdłuższy palindrom

Szablon rozwiązania: `zad2k.py`

Dany jest ciąg liter S. Proszę znaleźć taki SPÓJNY fragment tego podciągu, który będzie najdłuższym możliwym palindromem.

Algorytm należy zaimplementować jako funkcję postaci:

```
def palindrom( S ):
  …
```

która przyjmuje ciąg S i zwraca ten najdłuższy palindrom.

Przykład. Dla ciągu:

```
aacaccabcc
```

Wynikiem jest ciąg **acca**

Testowanie Rozwiązań

Żeby przetestować rozwiązania należy wykonać polecenie: `python3 zad2k.py`

Zadanie 3 - Najmniejsza k-ładna suma

Szablon rozwiązania: `zad3k.py`

Dla każdego ciągu n liczb możemy obliczyć k-ładną sumę (Zakładamy, że k <= n). Poprzez k-ładną sumę rozumiemy minimalną sumę pewnych liczb wybranych w ten sposób, że z każdych k kolejnych elementów wybraliśmy przynajmniej jeden z nich (w szczególności oznacza to, że dla k=1 musimy wybrać wszystkie elementy, a dla k=n wystarczy wybrać jeden, najmniejszy z nich). Proszę napisać algorytm, który dla zadanej tablicy liczb naturalnych oraz wartości k oblicza k-ładną sumę.

Algorytm należy zaimplementować jako funkcję postaci:

```
def ksuma( T, k ):
  …
```

która przyjmuje tablicę liczb naturalnych T = [$a_1, a_2, \ldots, a_n$] oraz liczbę naturalną k.

Przykład. Dla tablicy:

`[1, 2, 3, 4, 6, 15, 8, 7]` oraz `k = 4`

Wynikiem jest liczba **7**

Testowanie Rozwiązań

Żeby przetestować rozwiązania należy wykonać polecenie: `python3 zad3k.py`

Zadanie 4 - Ścieżka Falisza

Szablon rozwiązania: `zad4k.py`

Dana jest mapa wyrażona poprzez tablicę dwuwymiarową wymiarów N x N zawierająca liczby naturalne. Król Falisz znajduje się na polu (0,0) tej tablicy. Jego celem jest dojście do pola (n-1, n-1) i w trakcie tego procesu oblanie jak najmniejszej liczby studentów (każde pole tablicy wyraża ilość studentów, która zostanie oblana, gdy król przejdzie przez to pole). Ze względu na regulamin studiów Falisz może poruszać się jedynie o 1 pole w prawo lub w dół. Proszę napisać algorytm, który określi jaka jest minimalna liczba studentów, która zostanie oblana aby król doszedł do celu. Dla ułatwienia zadania pola (0, 0) oraz (n-1, n-1) przyjmują stałą wartość 0.

Algorytm należy zaimplementować jako funkcję postaci:

```
def falisz( T ):
  …
```

która przyjmuje dwuwymiarową tablicę liczb naturalnych T i zwraca liczbę będącą minimalną ilością studentów, których król musi oblać.

Przykład. Dla tablicy:

```
T = [
    [0, 5, 4, 3],
    [2, 1, 3, 2],
    [8, 2, 5, 1],
    [4, 3, 2, 0]
]
```

Wynikiem jest liczba **9**

Testowanie Rozwiązań

Żeby przetestować rozwiązania należy wykonać polecenie: `python3 zad4k.py`

Zadanie 5 - Ograj Garka

Szablon rozwiązania: `zad5k.py`

Dana jest talia N kart wyrażona poprzez tablicę A liczb naturalnych zawierającą wartości tych kart. Można przyjąć, że talia posiada parzystą ilość kart. Karty zostały rozłożone na bardzo szerokim stole w kolejności pojawiania się w tablicy. Dziekan poinformował Cię, że podwyższy Ci ocenę z WDI o pół stopnia, jeżeli wygrasz z nim w pewną grę, polegającą na braniu kart z jednego lub drugiego końca stołu na zmianę. Zakładając, że zaczynasz rozgrywkę, musisz znaleźć jaką maksymalnie sumę wartości kart uda Ci się uzyskać. Jednak, co ważne, musisz przyjąć, że dziekan jest osobą bardzo inteligentną i także będzie grał w 100% na tyle optymalnie, na tyle to możliwe. Aby nie oddawać losu w ręce szczęścia postanowiłeś, że napiszesz program, który zagwarantuje Ci wygraną (lub remis). Twój algorytm powinien powiedzieć Ci, jaka jest maksymalna suma wartości kart, którą masz szansę zdobyć grając z Garkiem.

Algorytm należy zaimplementować jako funkcję postaci:

```
def garek( A ):
  …
```

która przyjmuje tablicę liczb naturalnych T i zwraca liczbę będącą maksymalną możliwą do uzyskania sumą wartości kart.

Przykład. Dla tablicy:

```
T = [8, 15, 3, 7]
```

Wynikiem jest liczba **22**

Testowanie Rozwiązań

Żeby przetestować rozwiązania należy wykonać polecenie: `python3 zad5k.py`

Zadanie 6 - Hasło do laptopa

Szablon rozwiązania: `zad6k.py`

Podczas Twoich praktyk zawodowych w biurze śledczym otrzymałeś zadanie dostania się do pewnego zabezpieczonego hasłem laptopa. Jedyną podpowiedzią jaką zostawił przestępca był pewien ciąg cyfr wyrażony jako string S. Odkryto już, że w rzeczywistości podpowiedź pozostawiona przez przestępcę była pewną tajną wiadomością, która została zakodowana poprzez zamienienie liter na znaki (tak np. A = 1, B = 2, …, Z = 26) oraz, że hasło ustawione przez przestępcę to tak naprawdę liczba wyrażająca całkowitą liczbę różnych wiadomości, które mogą ukrywać się pod zakodowanym ciągiem. Twoim zadaniem jest napisanie algorytmu, który zwróci poprawne hasło niezbędne do zalogowania się do laptopa. Możesz przyjąć, że pusty ciąg ma tylko 1 rozwiązanie, a niepoprawne wiadomości 0 rozwiązań (przez niepoprawne można uznać np. takie, które posiadają dwa zera pod rząd, z których nie da się odczytać żadnej litery)

Algorytm należy zaimplementować jako funkcję postaci:

```
def haslo( S ):
  …
```

która przyjmuje string S i zwraca liczbę będącą poprawnym hasłem do laptopa.

Przykład. Dla ciągu znaków:

```
S = "123"
```

Wynikiem jest liczba **3** ponieważ zaszyfrowana wiadomość "123" może zostać zakodowane jako "ABC" (123), "LC" (12 3) lub "AW" (1 23).

Testowanie Rozwiązań

Żeby przetestować rozwiązania należy wykonać polecenie: `python3 zad6k.py`