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`

Zadanie 7 - Oszczędny ogrodnik

Szablon rozwiązania: `zad7k.py`

W sadzie pewnego oszczędnego ogrodnika rosły drzewa owocowe. Jednak ze względu na brak wystarczającej ilości funduszy, nie ma on możliwości podlania wszystkich z nich. Musi wybrać, które drzewa podlać, aby ze sprzedaży owoców z nich zebranych osiągnąć jak największy przychód. (Aby podlać dane drzewo musimy podlać cały jego korzeń, tj. wszystkie jego fragmenty posiadające przynajmniej jeden wspólny bok). Dana jest tablica dwuwymiarowa T o wymiarach NxM która zawiera informacje o tym, czy na danej "współrzędnej" znajduje się korzeń jakiegoś drzewa i jeżeli tak, to ile litrów wody wymaga, aby został poprawnie podlany. Pierwsza współrzędna tablicy T określa głębokość, a druga lokalizację. Ze względów logistycznych ogrodnik posiada księgę, w której zapisane są lokalizacje wszystkich drzew w sadzie. Wyrażona jest poprzez tablicę liczb naturalnych D. Przykładowo, jeżeli D[i] = x, oznacza to, że w punkcie T[0][x] będzie znajdował się fragment korzenia drzewa i-tego. Można założyć, że na głębokości "zerowej" każde drzewo posiada tylko jeden fragment korzenia, oraz, że żadne dwa drzewa nie mają wspólnego korzenia. Księgowa ogrodnika przygotowała także zbiór (wyrażony tablicą liczb naturalnych Z) potencjalnych przychodów, które może osiągnąć ze zbiórki owoców (Tak, że dla drzewa w lokalizacji D[i], potencjalny przychód wynosi Z[i]). Proszę napisać algorytm, który zwróci maksymalny przychód, który ogrodnik może osiągnąć ze zbiorów, zakładając, że posiada on tylko l litrów wody, aby podlać swój ogród.

Algorytm należy zaimplementować jako funkcję postaci:

```
def ogrodnik( T, D, Z, l ):
  …
```

która przyjmuje tablicę dwuwymiarową współrzędnych T, tablicę lokalizacji drzew D, tablicę potencjalnych zysków Z oraz limit litrów wody l.

Przykład. Dla danych:

```
D = [4, 9, 12, 16]
Z = [13, 11, 15, 4]
l = 32
```

Oraz następującej wizualizacji tablicy T (gdzie puste pola to zera, czyli brak korzenia)

				1					5			1				4			
				2					6			2				1			
				1				3	1			2	2	2		2	4	2	
			1	2			1	4	6		2	1	3			3	1		

Wynikiem jest liczba **28**

Testowanie Rozwiązań

Żeby przetestować rozwiązania należy wykonać polecenie: `python3 zad7k.py`

Zadanie 8 - Zepsuty wyświetlacz

Szablon rozwiązania: `zad8k.py`

W pewnym mieście zepsuły się wyświetlacze pokazujące nazwy przystanków autobusowych (zakładamy, że są one jednym słowem, bez spacji). Jako, że informatycy nie mogli sobie poradzić z problemem, postanowiono, że wszystkie literówki zostaną poprawione ręcznie przez pracowników. Pracownik może naprawiać napis poprzez dodawanie do niego liter, usuwanie z niego liter lub zamienianie istniejących liter na inne. Aby zoptymalizować swoją pracę musi wiedzieć, jaką najmniejszą ilość operacji musi wykonać, aby naprawić wyświetlacz. Napisz algorytm, który zwróci minimalną liczbę operacji, która musi zostać wykonana przez pracownika w celu naprawienia wyświetlacza.

Algorytm należy zaimplementować jako funkcję postaci:

```
def napraw( s, t ):
  …
```

która przyjmuje ciąg znaków s będący błędną nazwą przystanku na wyświetlaczu, oraz ciąg znaków t będący poprawna nazwą przystanku (do której uzyskania dążymy)

Przykład. Dla danych:

```
s = swidry
t = kawiory
```

Wynikiem jest liczba **3**

Testowanie Rozwiązań

Żeby przetestować rozwiązania należy wykonać polecenie: `python3 zad8k.py`

Zadanie 9 - Ładowanie promu (ze zwracaniem)

Szablon rozwiązania: `zad9k.py`

Dana jest tablica P z długościami samochodów (w umownej jednostce) które oczekują w kolejce, aby wjechać na prom. Pierwszy oczekujący samochód znajduje się pod zerowym indeksem. Prom ma dwa pokłady (górny i dolny) o długościach odpowiednio wyrażonych jako g i d. Zakładamy, że pojazdy mogą parkować "zderzak w zderzak" tj. bez zachowania odstępów między sobą. Każdy pojazd może wjechać na dowolnie wybrany z pokładów, jednak nie może wyprzedzić poprzedzających go samochodów. Niestety nie zawsze znajdzie się miejsce dla kolejnego samochodu. W takiej sytuacji kierowca ostatniego pojazdu, któremu udało się wjechać na prom, zgodnie z tradycją musi zamknąć wjazd oraz sporządzić listę obecności pojazdów TYLKO na swoim pokładzie w kolejności wjazdu pojazdów na ten pokład (W szczególności kierowca ten będzie ostatnim na liście obecności). Jako informatyk pracujący w biurze portowym, zostałeś poproszony o napisanie programu, który wskaże jak należy wpuszczać samochody na prom (tj. który samochód skierować na który pokład), aby ich łączna ilość na promie była jak największa.

Algorytm należy zaimplementować jako funkcję postaci:

```
def prom( P, g, d ):
  …
```

która przyjmuje tablicę długości pojazdów P = [$p_1, p_2, \ldots, p_n$] oraz długości pokładów g oraz d, a zwraca posortowaną tablicę indeksów pojazdów znajdujących się na liście obecności.

Przykład. Dla danych:

```
T = [5, 6, 1, 3, 2, 4, 3, 5]
l1 = 8
l2 = 10
```

Wynikiem jest między innymi tablica **[1, 4]**
Uw. Może być wiele poprawnych wyników

Testowanie Rozwiązań

Żeby przetestować rozwiązania należy wykonać polecenie: `python3 zad9k.py`

Zadanie 10 - Sklep z dywanami

Szablon rozwiązania: `zad10k.py`

Pewien przedsiębiorca potrzebuje zamówić do swojej firmy dywany o łącznym polu powierzchni wynoszącym N metrów kwadratowych. Nie martwi się on jakich będą one wymiarów, ponieważ może je w dowolny sposób przycinać i łączyć. Aczkolwiek sklep, w którym chce je kupić, sprzedaje tylko kwadratowe dywany o boku długości wyrażoną liczbą naturalną określającą długość w metrach. Koszt zapakowania każdego dywanu jest stały niezależnie od jego wielkości. Ze względów podatkowych przedsiębiorca potrzebuje zminimalizować łączny koszt zapakowania dywanów, które zakupi, jednocześnie dbając o środowisko nie może dopuścić, żeby jakikolwiek fragment dywanu się zmarnował. Twoim zadaniem jako jego pracownika jest stworzenie listy wymiarów dywanów (wyrażonych jako długość ich boku w metrach), które przedsiębiorca musi zakupić.

Algorytm należy zaimplementować jako funkcję postaci:

```
def dywany( N ):
  …
```

która przyjmuje wymagane pole powierzchni dywanów N w metrach kwadratowych, a zwraca tablicę długości boków dywanów, które trzeba kupić.

Przykład. Dla danych:

```
N = 6
```

Wynikiem jest np. tablica **[1, 1, 2]**

Testowanie Rozwiązań

Żeby przetestować rozwiązania należy wykonać polecenie: `python3 zad10k.py`

Zadanie 11 - Kontenerowiec

Szablon rozwiązania: `zad11k.py`

W porcie na odbiór oczekuje n kontenerów z towarem. Waga każdego kontenera jest znana i zapisana w tablicy T (w kilogramach). Dwuczęściowy kontenerowiec, który przypłynął odebrać towar jest ogromny - na tylko jednej z jego części zmieściłyby się wszystkie oczekujące kontenery. Jednak ze względów technicznych, aby statek nie zatonął, w każdej z dwóch jego części musi znajdować się towar o tej samej łącznej wadze. Z tego względu władze portowe muszą zaopatrzyć statek w pewną ilość kilogramowych odważników, które pozwolą na wyrównanie wagi w obydwu jego częściach. Odważniki te jednak są drogie, więc zależy im na tym, aby użyć ich jak najmniej. Twoim zadaniem jako programisty jest napisanie programu, który policzy, jaka jest ta najmniejsza możliwa liczba odważników.

Algorytm należy zaimplementować jako funkcję postaci:

```
def kontenerowiec( T ):
  …
```

która przyjmuje tablicę wag kontenerów T i zwraca najmniejszą konieczną liczbę odważników, które trzeba umieścić na statku.

Przykład. Dla danych:

```
T = [1, 6, 5, 11]
```

Wynikiem jest liczba **1**

Testowanie Rozwiązań

Żeby przetestować rozwiązania należy wykonać polecenie: `python3 zad11k.py`