[6pkt.] Zadanie 1.

Szablon rozwiązania:	zad1.py
Pierwszy próg złożoności:	$O(n^2)$
Drugi próg złożoności:	$O(n \log n)$

Mówimy, że ciąg liczb jest typu MR jeśli najpierw jest ściśle malejący a potem ściśle rosnący, albo jeśli jest tylko ściśle malejący lub tylko ściśle rosnący. Proszę zaimplementować funkcję

```
def mr( X )
```

która mając na wejściu ciąg liczb $X = [x_0, \ldots, x_{n-1}]$ zwraca jeden z jego najdłuższych podciągów typu MR.

Przykład. Dla wejścia $[4, 10, 5, 1, 8, 2, 3, 4]$ wynikiem jest $[10, 5, 1, 2, 3, 4]$. Dla wejścia $[1, 10, 5]$ poprawnymi wynikami są zarówno $[1, 10]$, $[1, 5]$, jak i $[10, 5]$.

[6pkt.] Zadanie 3.

Szablon rozwiązania:	zad3.py
Pierwszy próg złożoności:	złożoność czasowa $O(n + m)$
Drugi próg złożoności:	złożoność czasowa $O(m \log n)$ oraz pamięciowa $O(1)$

Dane jest pełne drzewo binarne T zawierające n wierzchołków. Każdy węzeł drzewa zawiera klucz będący liczbą całkowitą. Węzły drzewa numerujemy kolejnymi liczbami naturalnymi w ten sposób, że korzeń ma nr 1, jego synowie mają numery 2 i 3, następny poziom, od lewej do prawej, ma numery 4,5,6,7, itd. Dany jest ciąg X zawierający m liczb naturalnych ze zbioru $\{1, \ldots, n\}$. Należy założyć, że m jest istotnie mniejsze niż n. Proszę zaimplementować funkcję:

```
def maxim( T, X )
```

która zwraca maksymalny klucz spośród węzłów drzewa T o numerach wymienionych w X.

Funkcja powinna być możliwie jak najszybsza - wychodząc z założenia że $m << n$, i powinna działać na stałej pamięci (poza pamięcią potrzebną na przechowywanie danych wejściowych). Proszę oszacować złożoność czasową algorytmu.

Reprezentacja drzewa. Drzewo reprezentowane jest przy pomocy węzłów typu `Node`:

```
class Node:
  def __init__( self ):
    self.left   = None  # lewe poddrzewo
    self.right  = None  # prawe poddrzewo
    self.parent = None  # rodzic drzewa jeśli istnieje
    self.key    = None  # klucz
```

Przykład. Rozważmy drzewo, w którym klucze warstwami drzewa są umieszczone tak:

```
    5
  2 3
1 0 8 15
```

Niech $X = [3, 6, 4]$. W takim razie funkcja `maxim` powinna zwrócić wartość 8.

[6pkt.] Zadanie 2.

Szablon rozwiązania:	zad2.py
Pierwszy próg złożoności:	$O(dn \log n)$, gdzie n to liczba sekwencji, zaś d do długość pojedynczej sekwencji.
Drugi próg złożoności:	$O(D)$, gdzie D to sumaryczna długość wszystkich sekwencji.

Dana jest lista L parami różnych napisów składających się z symboli 0, 1. Mówimy, że pewien napis s jest fajny, jeśli jest prefiksem co najmniej dwóch napisów z L (przy czym jeśli w L znajduje się napis identyczny z s, to napis s wciąż traktujemy jako jego prefiks). Dalej, mówimy że napis s jest bardzo fajny, jeśli jest fajny a zarazem żadne jego rozszerzenie (poprzez dodanie dowolnego symbolu na końcu) nie jest napisem fajnym.

Zaproponuj, uzasadnij poprawność i zaimplementuj algorytm, który otrzymuje listę napisów L (składających się z zer i jedynek) i zwraca wszystkie bardzo fajne napisy dla tej listy. Algorytm powinien być zaimplementowany jako funkcja postaci:

```
def double_prefix(L):
    ...
```

gdzie `L` to lista zawierająca wejściowe napisy (jako napisy w języku Python). Funkcja powinna zwrócić listę prefiksów spełniających warunki zadania (również jako listę napisów języka Python). Prefiksy można zwrócić w dowolnej kolejności.

Przykład. Dla wejścia `['0100', '0110', '1010', '1']` prawidłowym wynikiem jest dowolna permutacja listy: `['01', '1']`.

Napisy w języku Python. Python umożliwia łatwe iterowanie po elementach napisu:

```
s = '0111'
for znak in s:
  print(znak)
```

Równie łatwo można dodawać znaki do napisu:

```
s = '0111'
s += '0'
inny_s = '' + '1'
```