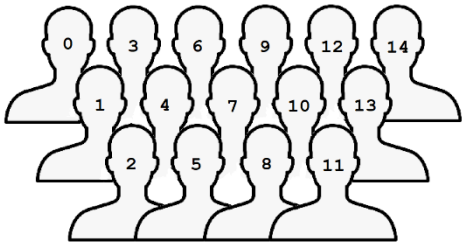
Szablon rozwiązania	`egzP2a.py`
Złożoność akceptowalna (1.5pkt):	O(*n*log*n*)
Złożoność wzorcowa (+2.5pkt):	O(*nm*), gdzie *n* to liczba osób, a *m* to ilość rzędów (*m* < log*n*)

W tym zadaniu nie wolno korzystać z wbudowanych funkcji sortowania.



Na zakończenie roku akademickiego Dziekan Gajęcki zdecydował, że na pamiątkę do powieszenia na ścianie zrobi zdjęcie grupowe pewnej grupy zainteresowanych studentów z I roku informatyki. Jednocześnie chciał, aby na zdjęciu wszyscy byli widoczni (Zakładamy, że osoba jest widoczna, jeżeli **w całym rzędzie pod nią** nie stoi żadna osoba wyższa od niej). Numeracja miejsc przedstawiona jest na rysunku (tj. zaczynamy numerację od lewego górnego rogu idąc „kolumnami" po skosie w dół, po dojściu do ostatniego rzędu wracamy do kolejnej osoby w górnym rzędzie). Jako, że dziekanowi zależy na ładnym zdjęciu, musi wyglądać w sposób analogiczny do przedstawionego na rysunku, tj. w każdym kolejnym (wyższym) rzędzie musi być o jedna osoba więcej. Aby wszystko poszło sprawnie, dzień wcześniej wysłał listę studentów starościnom i poprosił je o poprawne przypisanie im miejsc (czyli przestawienie ich w tablicy – zakładamy, że indeksy w tablicy to miejsca na zdjęciu)

W ramach zadania należy zaimplementować funkcję:

```
def zdjecie(T, m, k)
```

która edytuje w odpowiedni sposób przekazaną tablicę T i zwraca wartość None.

- **T** jest tablicą studentów, każdy student wyrażony jest w postaci krotki (*alb, w*), gdzie alb to numer indeksu, a w to wzrost studenta.

- **m** to ilość rzędów, a **k** to ilość osób znajdujących się w najniższym rzędzie

Rozważmy następujące dane

```
m = 2 #Ilość rzędów
k = 2 #Ilość osób w najniższym rzędzie

T = [ (1001, 154),(1002, 176),(1003, 189),(1004, 165),(1005, 162) ]
      #I rząd     #II rząd    #I rząd     #II rząd    #I rząd
```

Wywołanie `zdjecie(T, m, k)` powinno zwrócić wartość None oraz poprawnie edytować tablicę T np. w następujący sposób (może być **wiele poprawnych rozwiązań**):

```
T = [ (1002, 176),(1001, 154),(1003, 189),(1005, 162),(1004, 165) ]
      #I rząd     #II rząd    #I rząd     #II rząd    #I rząd
```

Szablon rozwiązania	`egzP2b.py`
Złożoność akceptowalna (1.5pkt):	$O(n^2 \log n + qm \log n)$
Złożoność dobra (+1.5pkt):	$O(nm + qm \log n)$
Złożoność wzorcowa (+1pkt):	$O(nm + qm)$, gdzie n określa liczbę słów w liście szyfrów, m to długość najdłuższego szyfru w liście szyfrów (m < n), a q to liczba słów binarnych w raporcie

Adam dostał za zadanie kontrolę informacji przesyłanych w państwowym intranecie. Praca przez lata była bardzo prosta, jednakże pewnej nocy zauważył serię podejrzanych ciągów bitowych. W związku z tym rozpoczął zbieranie wszystkich wiadomości, aby móc je odkodować. Zadanie wydawało się niesamowicie proste, ponieważ szpiedzy już dawno przechwycili szyfr, więc wystarczyłoby przetłumaczyć wszystkie wiadomości i wysłać raport. Niestety okazało się, że maszyna służąca do przekazywania raportów się popsuła i zamiast całego słowa pokazuje tylko sufiks tego słowa. Adam nie ma czasu do stracenia, więc musi wymyśleć na szybko algorytm, który obliczy liczbę wiadomości, które mogły zostać zakodowane w taki sposób jak w raporcie, aby wiedzieć czy będzie w stanie je wszystkie sprawdzić do następnego poniedziałku, czy może jednak liczba będzie tak ogromna, że musi zgłosić sprawę przełożonym. Twoim zadaniem jest napisać algorytm, który dla listy szyfrów **D** oraz raportu **Q** zwróci logarytm dziesiętny z liczby potencjalnych wiadomości w raporcie. Adam będzie zadowolony z programu, jeżeli Twoja odpowiedź będzie się różnić max. o 0.01 od rzeczywistego wyniku.

W ramach zadania należy zaimplementować funkcję:

```
def kryptograf( D, Q )
```

która obliczy i zwróci logarytm dziesiętny z maksymalnej możliwej liczby wiadomości

- W zadaniu do obliczenia logarytmu dziesiętnego należy użyć funkcji **log10** z biblioteki **math**. Obliczanie wyniku inaczej może spowodować rozbieżności z poprawnymi odpowiedziami.

- Porównanie dwóch ciągów znaków o długościach **x** i **y** ma złożoność O(min(**x**, **y**))

- Niepusty sufiks słowa to spójny fragment tego słowa zawierający ostatnią literkę. Pustym sufiksem każdego słowa jest ''. Przykładowo, wszystkimi sufiksami słowa '**Adam'** są: 'Adam', 'dam', 'am', 'm' oraz ''.

Rozważmy następujące dane

```
D = ["1100", "100", "0", "1111", "1101"]
Q = ["", "1", "11", "0", "1101"]
```

Wywołanie `kryptograf( D, Q )` powinno zwrócić ok. **1.47**
- Pierwsze słowo w Q jest puste, zatem może być sufiksem każdego słowa z D (**5** możliwości)
- Drugie słowo w Q to '**1**' i są dwa słowa, których może być sufiksem ('111**1**', '110**1**')
- Trzecie słowo kończy się na '**11**', więc może być jedynie sufiksem słowa '11**11**'
- Czwarte słowo kończy się na '**0**', więc może być to '110**0**', '10**0**' lub '**0**' (**3** możliwości)
- Piąte słowo nie zostało ucięte i musi być to po prostu '**1101**'

Daje to **5** * **2** * **1** * **3** * **1** = **30**. Logarytm dziesiętny z 30 wynosi ok. **1.47**

Podpowiedź. W rozwiązaniu na złożoność wzorcową proszę rozważyć użycie **BST**