Warunki

1. Zadania można wysyłać, jeśli ktoś ma najwyżej 2 plusy (licząc od 18 III).
2. W implementacjach można korzystać tylko z elementarnych konstrukcji Python'a (funkcje, instrukcje warunkowe, pętle, `range`, klasy użyte do definiowania struktur danych, wbudowana funkcja sortująca, itp.). **Nie wolno korzystać ze słowników i zbiorów.**
3. Rozwiązania muszą być efektywne obliczeniowo (także w zadaniach, w których nie podajemy wprost ograniczenia na złożoność obliczeniową). Zadania o zbyt wysokiej złożoności będą oceniane na brak plusa.
4. **Wolno omawiać zadania (w tym pomysły na implementację), ale wyłącznie na forum w systmie UPEL. Nie wolno wymieniać kodu realizującego fragmenty algorytmu (ale wolno odpowiadać na pytania postaci "jak zrealizować tablicę dwuwymiarową" itp.).**

Zadanie 1 (Kosztowna szachownica)

Dana jest szachownica o wymiarach $n \times n$. Każde pole (i, j) ma koszt (liczbę ze zbioru $\{1, \ldots, 5\}$) umieszczony w tablicy A (na polu $A[j][i]$). W lewym górnym rogu szachownicy (na pozycji $(0, 0)$ stoi król, którego zadaniem jest przejąść do prawego dolnego rogu, przechodząc po polach o minimalnym sumarycznym koszcie (jeśli król stoi na polu (i,j) to ponosi koszt $A[j][i]$; tak więc każda trasa zawiera koszt $A[0][0]$ i $A[n-1][n-1]$). Proszę zaimplementować funkcje `kings_path(A)`, która oblicza koszt sciezki króla. Funkcja powinna byc mozliwie jak najszybsza (w szczególności oczekujemy złożoności $O(n^2)$).

Państwa kod powinien mieć następującą postać (będzie uruchamiany; proszę nie usuwać fragmentu testującego; sprawdzający może także dołożyć swoje testy):

```
def kings_path( A ):
# policz koszt trasy

A = [[1,1,2],
     [5,1,3],
     [4,1,1]]
print( kings_path( A ) )  # wypisze 5
```

Zadanie 2 (Dwuwymiarowy problem plecakowy)

Proszę zaimplementować funkcję `knapsack2d(P, max_w, max_h)`, która oblicza maksymalną wartość plecaka w dwuwymiarowej wersji dyskretnego problemu plecakowego, określonego następująco. Mamy daną tablicę n trójek $P = [(v_0, w_0, h_0), \ldots (v_{n-1}, w_{n-1}, h_{n-1})]$, gdzie i-ta krotka ma następujące znaczenie: v_i to wartość i-go przemiotu, w_i to jego waga, a h_i to jego wysokość. Złodziej chce wybrać przedmioty o maksymalnej symarycznej wartości, których łączna waga nie przekracza danej liczby `max_w` oraz których łączna wysokość nie przekracza danej liczby `max_h` (przedmioty zapakowane są w kartony, które złodziej układa jeden na drugim).

Państwa kod powinien mieć następującą postać (będzie uruchamiany; proszę nie usuwać fragmentu testującego; sprawdzający może także dołożyć swoje testy):

```
def knapsack2d( V, max_w, max_h ):
  # tu proszę umieścić swoją implementację

  P = [(5,10,3), (7,8,12), (2,7,3)]
  print( knapsack2d( P, 16, 15 )   # wypisze 9
```