Warunki

1. Zadania można wysyłać, jeśl 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.

Zadanie 1 (wybór zajęć)

Dana jest n elementowa tablica $A = [(b_1, e_1), \ldots, (b_n, e_n)]$, gdzie każda para (b_i, e_i) oznacza zajęcia rozpoczynające się w chwili b_i i kończące w chwili e_i. Proszę zaimplementować funkcję `tasks(A)`, która zwraca ile maksymalnie zajęć można wybrać tak, by na siebie nie nachodziły. Można założyć, że wszystkie liczby w tablicy A są naturalne. Przedziały należy traktować jako otwarte, czyli np. zajęcia $(1, 3)$ oraz $(3, 5)$ nie nachodzą na siebie.

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

```
def tasks(A):
  # tu proszę umieścić swoją implementację

# elementarny test, powinien wypisać 2
print( tasks([ (0,10), (10,20), (5,15) ] )
```

Zadanie 2 (kody Huffmana)

Dana jest tablica n liczb naturalnych A. Liczba $A[i]$ mówi ile razy i-ty symbol pojawia się w tekście. Proszę zaimplementować funkcję `huffman_len(A)`, która oblicza ile bitów zajęłoby zapisanie tekstu składającego się właśnie z takiej liczby symboli, jeśli użytoby optymalnego kodu Huffmana. Funkcja powinna działać w czasie $O(n \log n)$. Podpowiedź: Może się przydać struktura kopca.

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

```
def huffman_len(A):
  # tu proszę umieścić swoją implementację

# elementarny test, powinien wypisać 2600
print( huffman_len([ 200, 700, 180, 120, 70, 30] )
```

Zadanie 3 (ciągły problem plecakowy)

Dana jest n elementowa tablica $A = [(P_1, W_1), \ldots, (P_n, W_n)]$ opisująca egzemplarz ciągłego problemu plecakowego; A opisuje dostępne płyny a k objętość plecaka (a raczej pojemnika; k jest podane w litrach). Dla i-go przedmiotu P_i oznacza jego wartość za wszystkie dostępne W_i litrów. Proszę zaimplementować funkcję `knapsack(A,k)`, która oblicza wartość najlepszego pojemnika, jaki można uzyskać.

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

```
def knapsack(A, k):
  # tu proszę umieścić swoją implementację

# elementarny test, powinien wypisać 12
print( tasks( [ (1,1), (10,2), (6,3) ], 3 )
```