## 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 (Krawędzie 0/1)

Proszę zaimplementować funkcję `path_cost(G,s,t)`, która oblicza koszt najtańszej ścieżki w grafie skierowanym $G$ z wierzchołka $s$ do wierzchołka $t$, gdzie każda krawędź ma koszt ze zbioru $\{0, 1\}$. Graf jest reprezentowany jako listy sąsiedztwa.

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

```
def path_cost( G, s, t ):
# policz koszt trasy
n = len(G)   # liczba wierzchołków grafu
G[i]         # lista informacji o wierzchołkach do których
             # jest bezpośrednia krawędź z wierzchołka i
# np. G[i] = [(7,0), (4,1), (8,1), (2,0)] oznacza, że z wierzchołka
# i są krawędzie do wierzchołków 7 (koszt 0), 4 (koszt 1),
# 8 (koszt 1) i 2 (koszt 0)

G = [[(1,0), (2,1)],
     [(3,1), (2,0)],
     [(3,0)],
     []]
print( path_cost( G, 0, 3 ) )  # wypisze 0
```

## Zadanie 2 (Liczenie ścieżek)

Proszę zaimplementować funkcję `count_shortest_paths(G,s,t)`, która oblicza ile różnych najkrótszych ścieżek prowadzi w grafie skierowanym $G$ z wierzchołka $s$ do wierzchołka $t$. Dwie ścieżki są różne jeśli różnią się choć jedną krawędzią. Graf $G$ jest reprezentowany macierzowo (`G[i][j] == True` jeśli istnieje krawędź z `i` do `j`; `G[i][j] = False` w przeciwnym wypadku).

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

```
def count_shortest_paths( G, s, t ):
  # tu proszę umieścić swoją implementację
```

```
G = [[False, True,  True,  False],
     [False, False, True,  True ],
     [False, False, False, True ],
     [False, False, False, False]]
print( count_shortest_paths( G, 0, 3 )   # wypisze 2
```