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 (Domknięcie przechodnie)

Domknięciem przechodnim skierowanego grafu $G = (V, E)$ nazywamy taki graf $G' = (V, E')$, że dla każdych dwóch wierzchołków $u, v \in V$ graf G' ma krawędź z u do v wtedy i tylko wtedy, gdy w G jest skierowana ścieżka z u do v. Proszę zaimplementować funkcję `tclosure( G )`, która na wejście otrzymuje graf skierowany w reprezentacji macierzowej (bez wag; $G[i][j]$ to wartość logiczna mówiąca czy istnieje krawędź z i do j) i zwraca graf będący domknięciem przechodnim G (w tej samej reprezentacji).

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

```
def tclosure( G ):
# policz domknięcie przechodnie G i je zwróc


G = [ [False, True , False],
      [False, False, True ],
      [False, False, False] ]
print( tclosure( G ) )  # wypisze
# [[False, True , True],
#  [False, False, True],
#  [False, False, False]]
```

Zadanie 2 (Algorytm Dijkstry)

Proszę zaimplementować funkcję `dijkstra( G, s )`, która znajduje najkrótsze ścieżki w grafie skierowanym `G` (representowanym listowo) z `s` do wszystkich innych osiągalnych wierzchołków. Funkcja zwraca tablicę obliczonych pól `parent`.

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

```
def dijkstra( G, s ):
# policz 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( dijkstra( G, 0 ) )  # wypisze [None,0,1,2]
```