1	2	3	4	5	6	7	8	Σ
+	-	+	+	+	-	-	-	—

ZADANIE 1.

Napisz rekurencyjne funkcje, które dla danego drzewa binarnego T obliczają:

Zakładam, że drzewa są pełne binarne (czyli albo ma dwoje dzieci, albo ani jednego)

(a) liczbę wierzchołków w T

```
# dupa

A[n][2] <- macierz sasiedztwa T, -1 jesli nie ma sasiadow

function cnt_vertices(v):
    if A[v][0] != -1:
        return cnt_vertices(A[v][0]) + cnt_vertices(A[v][1]) + 1
    else:
        return 1
```

(b) maksymalną odległość między wierzchołkami w T.

```
# dupa

A[n][2] <- macierz sasiedztwa T, -1 jesli nie ma sasiadow

function max_dist(v, m):
    if A[v][0] != -1:
        left <- max_dist(A[v][0])
        right <- max_dist(A[v][1])
        max_fork = max(left[0], right[0], left[1] + right[1] + 2)
        max_down = max(left[1], right[1]) + 1
        return [max_fork, max_down]
    return [0, 0]

ret <- max_dist(0)
OUTPUT max(ret[0], ret[1])
```

ZADANIE 3.

Porządkiem topologicznym *wierzchołków acyklicznego digrafu* $G = (V, E)$ *nazywamy taki liniowy porządek jego wierzchołków, w którym początek każdej krawędzi występuje przed jej końcem. Jeśli wierzchołki z* V *utożsamimy z początkowymi liczbami naturalnymi, to każdy ich porządek liniowy można opisać permutacją liczb* $1, \ldots, |V| = n$*; w szczególności pozwala to na porównanie leksykograficzne porządków.*

Ułóż algorytm, który dla danego acyklicznego digrafu znajduje pierwszy leksykograficznie porządek topologiczny.

```
A <- macierz sasiedztwa
IN <- tablica ilosci krawedzi wchodzacych do wierzcholka

ret <- []
golaski <- [] kolejka piorytetowa
```

```
i <- 0

# sprawdzam potencjalne poczatki mojej permutacji i pamietam je w
   kolejnosci rosnacej
while i < n:
    if IN[i] == 0:
        golaski.insert-node(i)

# dopoki mam potencjalne wierzcholki do wstawienia
while len(golaski) > 0:
    m <- find-min(golaski)
    ret.append(m)
    golaski.deletemin()

    # obcinam golaska od jego sasiadow
    for i in A[m]:
        IN[i]--
        if IN[i] == 0:
            golaski.push_back(i)

OUTPUT ret
```

ZADANIE 4.

Niech u i v będą dwoma wierzchołkami w grafi nieskierowanym $G = (V, E; c)$, gdzie $c : E \to \mathbb{R}_+$ jest funkcją wagową. Mówimy, że droga z $u = u_1, \ldots, u_{k-1}, u_k = v$ z u do v jest sensowna, jeśli dla każdego $i = 2, \ldots, k$ istnieje droga z u_i do v krótsza od każdej drogi z u_{i-1} do v (przez długość drogi rozumiemy sumę wag jej krawędzi).

Ułóż algorytm, który dla danego G oraz wierzchołków u i v wyznaczy liczbę sensownych dróg z u do v.

```
A <- tablica somsiadow

D <- robimy Dijkstre, dostajemy tablice

i <- 0

S <- [[] * n]

while i < n:
    for j in A[i]:
        if D[i] > D[j]:
            S[i].push_back(j)
        else if D[j] > D[i]:
            S[j].push_back(i)

P <- topo sort <3

Dupa <- puste n * []
Dupa[u] <- 1

for i in P:
```

```
    for j in S[i]:
        Dupa[j] += Dupa[i]

OUTPUT Dupa[v]
```

ZADANIE 5.

Ułóż algorytm, który dla zadanego acyklicznego grafy skierowanego G znajduje długość najdłuższej drogi w G. Następnie zmodyfikuj swój algorytm tak, by wypisywał drogę o największej długości (jeśli jest kilka takich dróg, to Twój algorytm powinien wypisać dowolną z nich).

```
A <- lista sasiedztwa
IN <- stworzona przy wczytywaniu lista

golaski <- []

MAX_DROGA <- tablica n razy 0

i <- 0
while i < n:
    if IN[i] == 0:
        golaski.append(i)

while len(golaski) > 0:
    m <- golaski[len(golaski)]
    golaski.pop_back()

    for i in A[m]:
        IN[i]--
        if IN[i] == 0:
            golaski.push_back(i)

        # tutaj maksymalna droga do i to jest albo to co bylo, albo droga do m powiekszona o 1?
        MAX_DROGA[i] = max(MAX_DROGA[m] + 1, MAX_DROGA[i])

OUTPUT max(MAX_DROGA)
```