

1	2	3	4	5	6	7	8	$\Sigma$
+	-	+	+	+	-	-	-	-

## 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  $T$

```

1 # dupa
2
3 A[n][2] <- macierz sasiedztwa T, -1 jesli nie ma sasiadow
4
5 function cnt_vertices(v):
6     if A[v][0] != -1:
7         return cnt_vertices(A[v][0]) + cnt_vertices(A[v][1]) + 1
8     else:
9         return 1

```

(b) maksymalną odległość między wierzchołkami w  $T$ .

```

1 # dupa
2
3 A[n][2] <- macierz sasiedztwa T, -1 jesli nie ma sasiadow
4
5 function max_dist(v, m):
6     if A[v][0] != -1:
7         left <- max_dist(A[v][0])
8         right <- max_dist(A[v][1])
9         max_fork = max(left[0], right[0], left[1] + right[1] + 2)
10        max_down = max(left[1], right[1]) + 1
11        return [max_fork, max_down]
12    return [0, 0]
13
14 ret <- max_dist(0)
15 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  $1, \dots, |V| = n$ ; w szczególności pozwala to na porównanie leksykograficzne porządków.

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

```

1 A <- macierz sasiedztwa
2 IN <- tablica ilosci krawedzi wchodzacych do wierzcholka
3
4 ret <- []
5 golaski <- [] kolejka priorytetowa

```

```

6
7 i <- 0
8
9 # sprawdzam potencjalne poczatki mojej permutacji i pamietam je w
  kolejnosci rosnacej
10 while i < n:
11     if IN[i] == 0:
12         golaski.insert-node(i)
13
14 # dopoki mam potencjalne wierzcholki do wstawienia
15 while len(golaski) > 0:
16     m <- find-min(golaski)
17     ret.append(m)
18     golaski.deletemin()
19
20     # obcinam golaska od jego sasiadow
21     for i in A[m]:
22         IN[i]--
23         if IN[i] == 0:
24             golaski.push_back(i)
25
26 OUTPUT ret

```

## ZADANIE 4.

Niech  $u$  i  $v$  będą dwoma wierzchołkami w grafie nieskierowanym  $G = (V, E; c)$ , gdzie  $c : E \rightarrow \mathbb{R}_+$  jest funkcją wagową. Mówimy, że droga z  $u = u_1, \dots, u_{k-1}, u_k = v$  z  $u$  do  $v$  jest sensowna, jeśli dla każdego  $i = 2, \dots, k$  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  $v$ .

```

1 A <- tablica somsiadow
2
3 D <- robimy Dijkstre , dostajemy tablice
4
5 i <- 0
6
7 S <- [[] * n]
8
9 while i < n:
10     for j in A[i]:
11         if D[i] > D[j]:
12             S[i].push_back(j)
13         else if D[j] > D[i]:
14             S[j].push_back(i)
15
16 P <- topo sort <3
17
18 Dupa <- puste n * []
19 Dupa[u] <- 1
20
21 for i in P:

```

```
22     for j in S[i]:
23         Dupa[j] += Dupa[i]
24
25 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).

```
1 A <- lista sasiedztwa
2 IN <- stworzona przy wczytywaniu lista
3
4 golaski <- []
5
6 MAX_DROGA <- tablica n razy 0
7
8 i <- 0
9 while i < n:
10     if IN[i] == 0:
11         golaski.append(i)
12
13 while len(golaski) > 0:
14     m <- golaski[len(golaski)]
15     golaski.pop_back()
16
17     for i in A[m]:
18         IN[i]--
19         if IN[i] == 0:
20             golaski.push_back(i)
21
22     # tutaj maksymalna droga do i to jest albo to co bylo, albo
23     # droga do m powiekszona o 1?
24     MAX_DROGA[i] = max(MAX_DROGA[m] + 1, MAX_DROGA[i])
25
26 OUTPUT max(MAX_DROGA)
```