# Feladat

Egy hierarchikusan felépülő társaságnak egy olyan tagja van, akinek nincsen felettese. Ha egy tag levelet küld egy másik tagnak, akkor a levél csak közvetlen felettesnek vagy közvetlen felettestől kerülhet át egy lépésben, így jutva el lépésenként a feladótól a címzettnek.

* Az első részfeladat alapján csak beosztottnak lehet továbbítani egy lépésben a levelet. Döntsük el, hogy két megadott tag közül melyik küldhet levelet a másiknak!
* A második részfeladatban a levél egy lépésben bármilyen irányba közlekedhet. Adjuk meg, hogy két tag között mennyi lépésben adható át egy levél!
* A harmadik részfeladatban adjuk meg, hogy a két tagnak külön-külön hány közvetlen és közvetett beosztottja van összesen!

# Eredeti feladat

<https://mester.inf.elte.hu:8181/faces/feladat.xhtml?jfwid=082a01a307c9aa976a0907310d88:11>

# Ötlet

A társaság tagjainak közvetlen beosztottjait egy tömböket tartalmazó tömbben tároljuk. A tömb i-edik eleme az i-edik ember közvetlen beosztottjait tartalmazza. Az eredeti feladat mintabemenete esetén ez a tömb az alábbi módon néz ki:

[[2, 3], [5], [4, 6, 7], [], [], [], []]

A részfeladatok megoldásához emlékezzünk a rekurzív adatszerkezetekről tanultakra! Segítségünkre lesz a rekurzív mélységi bejárás részbeni alkalmazása (mindegyik részfeladatban segít).

# Megoldás

1. Az első részfeladat megoldásához egy rekurzív függvényhívásra lesz szükségünk, melynek struktogramja a következő:

A képen képernyőkép látható

Automatikusan generált leírás

a paraméterként adott értékek a következők:

h: a tömb, amiben a közvetlen beosztottak tömbjei vannak  
x: a feladó  
y: a címzett

A result változóban fogjuk eltárolni, hogy x közvetlen és közvetett beosztottjai közt szerepel-e y.

X tömbjéből indulunk, és minden elemét indexül véve megvizsgáljuk a többi tömböt is, egészen addig, amíg valamelyik tömb eleme nem lesz egyenlő y-nal és van még olyan eleme a tömb x-edik elemének, amit még nem vizsgáltunk. Ha valamelyik elem egyenlő y-nal, akkor a result változót True-ra állítjuk. A ciklus lefutása után visszatérési értékül adjuk a result logikai változót, tehát megkapjuk, hogy y beosztottja-e x-nek.

1. A második részfeladat megoldásához két függvényre lesz szükségünk. Az egyik megadja, hogy külön-külön x-től és y-tól milyen útvonalon jutunk el a gyökérelemig. Ennek a struktogramja a következő:

A képen képernyőkép látható

Automatikusan generált leírás

Itt a bemeneti paraméterek a következők:

h: a tömb, amiben a közvetlen beosztottak tömbjei vannak  
a: az az elem, amitől az útvonalat keressük a gyökérelemig

A result tömbben tároljuk el az adott elemeket, amiken keresztül eljutunk a-tól a gyökérelemig. A tömb első eleme maga a, az utolsó elem pedig a gyökérelem (1) lesz majd. Ha a egyenlő a gyökérelemmel, akkor egyből visszaadjuk a tömböt, amelynek egyetlen eleme a gyökérelem (tehát a). Ha viszont a két elem nem egyezik, akkor ciklust indítunk, amely h tömbjein megy keresztül és keresi azt a tömböt, aminek eleme a (biztosan talál ilyet, mivel ha a nem egyenlő a gyökérelemmel, akkor lennie kell felettesének). Ha megtalálta a keresett tömböt, annak indexére szintén meghívja a függvényt, majd annak eredményét hozzáfűzi a result tömbhöz, majd végül ezt adja visszatérési értéknek.

Ezt követően a másik függvény, amire szükségünk van, meghívja a routeToRoot() függvényt x-re és y-ra is, ez után az alábbi struktogramnak megfelelő műveleteket hajtja végre velük:

A képen képernyőkép látható

Automatikusan generált leírás

A paraméterül adott értékek az alábbiakkal egyeznek meg:

h: a tömb, amiben a közvetlen beosztottak tömbjei vannak  
a: x  
b: y

Miután kimentettük a két útvonalat a gyökérelemig xRoute-ba és yRoute-ba, megkeressük az első olyan elemet, amit mindkettő tartalmaz, ezt a metszetet a section változóban tároljuk (biztos lesz metszet, legrosszabb esetben csak a gyökérelem). Ez után xRoute-ban addig megyünk az első elemtől, amíg az aktuális elem meg nem egyezik a section-nel (biztos találunk egyezést, hiszen a section-t xRoute elemei közül választottuk). Amíg el nem érünk a kívánt elemig, a kezdetben nullával egyenlő result értékét eggyel növeljük, így a ciklus végére megkapjuk azon résztömb hosszát, ami a section előtt van xRoute-ban. Ez után ugyanezt végrehajtjuk yRoute-ra is, ám itt a result értékét nem nullázzuk le, így a két eredmény összegét kapjuk végül, ami megegyezik a keresett útvonal hosszával is.

1. A harmadik részfeladatban szintén rekurzív függvényt fogunk alkalmazni. Ennek struktogramja a következő:

A képen képernyőkép látható

Automatikusan generált leírás

h: a tömb, amiben a közvetlen beosztottak tömbjei vannak  
a: Az elem, amelynek közvetlen és közvetett beosztottjainak számát keressük

A result értékét kezdetben nullára állítjuk, ebben fogjuk tárolni a beosztottak számát. Ezután végigmegyünk h a-adik tömbjén (tehát a közvetlen beosztottjain). Minden elem vizsgálatakor növeljük a result értékét eggyel, valamint annak az értékével, amit az adott elemre meghívott junorsOf() függvény ad. Amikor végigértünk az a-adik tömb összes elemén, visszaadjuk a result értékét.

# Implementáció

Python 3:

from sys import stdin, stdout

#######################################Beolvasásért felelős függvény

def readFromConsole():

inputNums = list(map(int, stdin.readline().split()))

N = inputNums[0]

X = inputNums[1]

Y = inputNums[2]

junior = []

for i in range(0, N):

junior.append([])

for i in range(1, N):

inputNum = int(stdin.readline())

junior[inputNum-1].append(i+1)

return [N,X,Y,junior]

#######################################Az első feladatért felelős függény

def onlyDown(h,x,y):

result = False

i = 0

while (i < len(h[x-1]) and not result):

if (h[x-1][i] == y-1):

result = True

else:

result = onlyDown(h, h[x-1][i], y)

i = i+1

return result

#######################################A harmadik feladatért felelős függény

def juniorsOf(h,a):

result = 0

for e in h[a-1]:

result = result + 1

result = result + juniorsOf(h, e)

return result

#######################################A második feladatért felelős függények

def routeToRoot(h,a):

result = [a]

if (a == 1):

return result

else:

i = 0

while (not a in h[i]):

i = i + 1

result = result + routeToRoot(h,i+1)

return result

def routeLength(h,a,b):

xRoute = routeToRoot(h,a)

yRoute = routeToRoot(h,b)

section = 0

i = 0

while (not (xRoute[i] in yRoute)):

i = i + 1

section = xRoute[i]

result = 0

i = 0

while (not xRoute[i] == section):

result = result + 1

i = i + 1

i = 0

while (not yRoute[i] == section):

result = result + 1

i = i + 1

return result

#######################################A főprogram függénye

def main():

parameters = readFromConsole()

N = parameters[0]

X = parameters[1]

Y = parameters[2]

junior = parameters[3]

#######################################1. feladat

if (onlyDown(junior, X, Y)):

stdout.write(str(1) + '\n')

elif (onlyDown(junior, Y, X)):

stdout.write(str(2) + '\n')

else:

stdout.write(str(3) + '\n')

#######################################2. feladat

stdout.write(str(routeLength(junior,X,Y)) + '\n')

#######################################3. feladat

stdout.write(str(juniorsOf(junior,X)) + " " + str(juniorsOf(junior,Y)))

main()