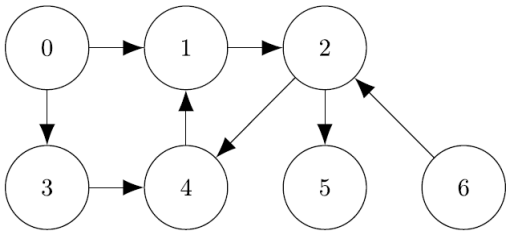
Devoir Surveillé 2

Durée : 1 heure 30 min

Calculatrice interdite

Exercice 0.1. *Soit $G := (S, A)$ un graphe où $S = [\![0; n-1]\!]$.*
On appelle liste d'adjacence le tableau g de longueur n tel que pour tout $i \in [\![0; n-1]\!]$, g_i est la liste des successeurs $j \in [\![0; n-1]\!]$ de i.
On considère le graphe suivant :



1. *Donner la liste d'adjacence correspondante (sous forme de liste de listes avec la syntaxe Python).*
   Par exemple :

```
>>> liste_adj[0]
[1, 3]
>>> liste_adj[2]
[4, 5]
```

2. *On rédige une fonction* ***parcourt(g,s)*** *qui reçoit la liste d'adjacence d'un graphe ainsi que l'indice $s \in [\![0; n-1]\!]$ d'un sommet et qui renseigne les sommets accessibles depuis le sommet s sous forme d'une liste de booléens.*

```
def parcourt(g,s):
    n=len(g)
    visite=[False for i in range(n)]
    pile=[s]
    while len(pile)!=0:
        x=pile.pop()
        if not visite[x]:
            visite[x]=True
            for y in g[x]:
                pile.append(y)
    return(visite)
```

   Par exemple :

```
>>> parcourt(liste_adj,0)
[True, True, True, True, True, True, False]
```

(a) *De quel type d'algorithme de parcourt de graphe s'agit-il ? Connaissez-vous une autre méthode de parcourt ? Expliquer la différence entre les deux méthodes.*

(b) *Expliquer le fonctionnement de cet algorithme en quelques lignes.*

3. *Rédiger une fonction* ***sommet(g,s)*** *qui renvoie la liste des sommets accessibles depuis le sommet s du graphe g (représenté par sa liste d'adjacence).*

   Par exemple :

```
>>> sommets(liste_adj,6)
[1, 2, 4, 5, 6]
```

4. *Rédiger une fonction* ***relies(g,s1,s2)*** *qui renvoie True si le sommet s2 est accessible depuis s1 en parcourant le graphe. Par exemple :*

```
>>> relies(liste_adj,0,3)
True
>>> relies(liste_adj,0,6)
False
```

5. *Un graphe non orienté est dit* ***connexe*** *si quels que soient les sommets s1 et s2, il existe une chaîne reliant s1 à s2.*

   (a) *Le graphe cité en exemple est-il connexe ?*

   (b) *Rédiger une fonction estconnexe(g) qui renvoie True si le graphe g représenté par sa liste d'adjacence est connexe, False sinon.*

Exercice 0.2. *On s'intéresse à n articles périssables représentés par une liste* $[0, 1, ..., n-1]$.
Pour chacun de ces articles, on dispose :

- *de son poids*
- *de son prix en euros*
- *du nombre de jours avant péremption*

Chacune de ses données est représentée sous forme de liste :

```
Articles=[0,1,2]
Poids=[2.3,4,1.5]
Prix=[2.5,5,8]
Jours=[6,5,4]
```

Par exemple, l'article 1 pèse 4 kg, coûte 5 euros et sera périmé dans 5 jours.

1. *Écrire une fonction* ***rentable(Poids, Prix)*** *qui reçoit en paramètres :*

- *une liste* ***Poids*** *des poids en kilos de n articles périssables*
- *une liste* ***Prix*** *des prix de ces articles*

et renvoie :

- *l'indice de l'article le moins cher au kilo.*
- *le prix au kilo de cet article*

2. *On rappelle le tri par insertion pour trier dans l'ordre croissant une liste T :*

```
def tri_insertion(T):
    for i in range(1,len(T)):
        j = i
        x=T[i]
        while j>0 and T[j-1]>x:
                T[j] = T[j-1]
                j = j-1
        T[j] = x
    return T
```

Écrire une fonction ***tri_articles(Poids)*** *qui renvoie la liste des poids triés dans l'ordre croissant avec pour chaque poiuds, l'article correspondant.*
Par exemple :

```
>>> tri_articles(Poids)
[[2, 1.5], [0, 2.3], [1, 4]]
```