EXERCICE 2 (6 points)

Cet exercice porte sur les graphes, la programmation, la structure de pile et l'algorithmique des graphes.

On s'intéresse à la fabrication de pain. La recette est fournie sous la forme de tâches à réaliser. Cette recette est réalisée par une personne seule.

(a) Préparer 500g de farine.
(b) Préparer 1/3 de litre d'eau (33cl).
(c) Préparer 1 c. à café de sel.
(d) Préparer 20g de levure de boulanger.
(e) Faire tiédir l'eau dans une casserole.
(f) Délayer la levure dans l'eau tiède.
(g) Laisser reposer la levure 5 minutes.
(h) Préparer un grand saladier.
(i) Verser la farine dans le saladier.
(j) Verser le sel dans le saladier.
(k) Mélanger la farine et le sel puis creuser un puits.
(l) Verser l'eau mélangée à la levure dans le puits.
(m) Pétrir jusqu'à obtenir une pâte homogène.
(n) Couvrir à l'aide d'un linge humide et laisser fermenter au moins 1h30.
(o) Disposer dans le fond du four un petit récipient contenant de l'eau.
(p) Préchauffer un four à 200 degrés Celsius.
(q) Fariner un plan de travail.
(r) Verser la pâte à pain sur le plan de travail.
(s) Pétrir rapidement la pâte à pain.
(t) Disposer la pâte dans un moule à cake.
(u) Mettre au four pour 15 à 20 minutes, arrêter le four et sortir le pain.

La figure 1 représente les différentes tâches et les dépendances entre ces tâches sous la forme d'un graphe. Chaque sommet du graphe représente une tâche à réaliser. Les dépendances entre les tâches sont représentées par les arcs entre les sommets.

Par exemple, il y a une flèche sur l'arc qui part du sommet d'étiquette (l) et qui atteint le sommet d'étiquette (m) car il faut avoir réalisé la tâche "*Verser l'eau mélangée à la levure dans le puits.*" (l) avant de pouvoir réaliser la tâche "*Pétrir jusqu'à obtenir une pâte homogène.*" (m).

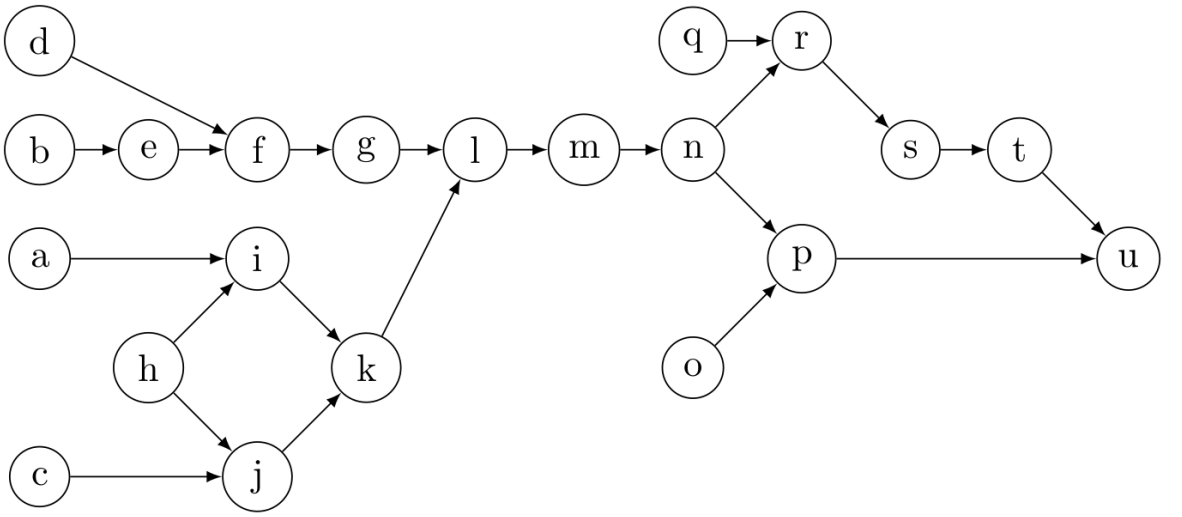

Figure 1. Recette du pain : tâches à effectuer avec leurs dépendances

1. Dire, sans justifier, s'il s'agit d'un graphe orienté ou non orienté.
2. D'après le graphe, dire s'il est possible d'effectuer les réalisations dans chacun des ordres suivants :
   - réaliser la tâche (f) puis la tâche (g)
   - réaliser la tâche (g) puis la tâche (f)
   - réaliser la tâche (i) puis la tâche (j)
   - réaliser la tâche (j) puis la tâche (i)
3. Donner toutes les tâches qu'il faut nécessairement avoir réalisées depuis le début pour pouvoir réaliser la tâche (k). Ne donner que les tâches nécessaires.
4. Indiquer, sans justifier, si le graphe de la Figure 1 contient un cycle.

Graphe de tâches

On s'intéresse désormais de manière plus générale à un graphe de tâches avec des dépendances.

Les sommets sont nommés par des indices. Comme précédemment, un arc orienté d'un sommet d'indice `i` à un sommet d'indice `j` signifie que la tâche représentée par le sommet d'indice `i` doit être réalisée avant la tâche représentée par le sommet d'indice `j`.

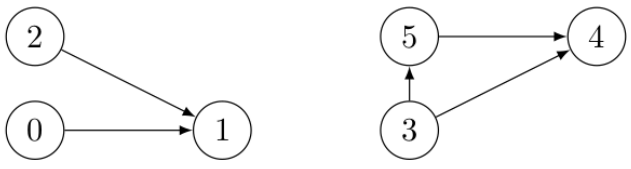

Figure 2. Exemple de graphe de dépendances entre 6 tâches

5. Déterminer un ordre permettant de réaliser toutes les tâches représentées dans le graphe de la Figure 2 en respectant les dépendances entre les tâches.

Voici une matrice d'adjacence d'un graphe écrite en langage Python et telle que si `M[i][j] = 1` alors il existe un arc qui va du sommet d'indice `i` au sommet d'indice `j`. Par exemple, `M[0][1] = 1` alors il existe un arc qui va du sommet d'indice `0` au sommet d'indice `1`.

```
M = [ [0, 1, 0, 0, 0],
      [0, 0, 1, 0, 0],
      [0, 0, 0, 1, 0],
      [0, 1, 0, 0, 1],
      [0, 0, 0, 0, 0] ]
```

6. Représenter le graphe associé à cette matrice d'adjacence. Les noms des sommets seront leurs indices.

7. Déterminer s'il est possible de trouver un ordre permettant de réaliser les tâches représentées par le graphe de la question 6 en respectant leurs dépendances. Si oui, donner l'ordre. Si non, expliquer pourquoi.

Voici le code Python d'une fonction `mystere`.

```
1  def mystere(graphe, s, n, ouverts, fermes, resultat):
2    """ Paramètres :
3        graphe    un graphe représenté par une matrice d'adjacence
4        s         l'indice d'un sommet du graphe
5        n         le nombre de sommets du graphe
6        ouverts   une liste de booléens permettant de savoir
7                  si le traitement d'un sommet a été commencé
8        fermes    une liste de booléens permettant de savoir
9                  si le traitement d'un sommet a été terminé
10       Retour :  False s'il y a eu un "problème", True sinon.
11       Le paramètre resultat sera modifié ultérieurement.
12   """
13   if ouverts[s]:
14       return False
15   if not fermes[s]:
16       ouverts[s] = True
17       for i in range(n):
18           if graphe[s][i] == 1:
19             val = mystere(graphe, i, n, ouverts, fermes,
                             resultat)
20             if not val:
21                 return False
22       ouverts[s] = False
23       fermes[s] = True
24       # ...
25   return True
```

8. En utilisant la matrice `M` donnée précédemment, déterminer si la variable `ok` vaut `True` ou `False` à l'issue des instructions suivantes :

```
1 n = len(M)
2 ouverts = [ False for i in range(n) ]
3 fermes = [ False for i in range(n) ]
4 ok = mystere(M, 1, n, ouverts, fermes, None)
```

Décrire précisément les appels effectués à la fonction `mystere` et les valeurs des tableaux `ouverts` et `fermes` lors de chaque appel. On pourra recopier et compléter le tableau ci-dessous.

Appel `mystere`	variable `ouverts`	variable `fermes`
Avant l'appel `mystere`	`[F,F,F,F,F]`	`[F,F,F,F,F]`
`mystere(M,1,5,[F,F,F,F,F], [F,F,F,F,F],None)`	`[F,T,F,F,F]`	`[F,F,F,F,F]`
`mystere(M,2,5,[F,T,F,F,F], [F,F,F,F,F],None)`	`[F,T,T,F,F]`	`[F,F,F,F,F]`
`...`		

9. De manière générale, expliquer dans quel cas cette fonction `mystere` renvoie `False`.

L'objectif est d'utiliser la fonction `mystere` pour écrire une fonction `ordre_realisation` qui, lorsque c'est possible, détermine l'ordre de réalisation des tâches d'un graphe donné par sa matrice d'adjacence en respectant les dépendances entre les tâches.

Une structure de données de pile est représentée par une classe `Pile` qui possède les méthodes suivantes :

- la méthode `estVide` qui renvoie `True` si la pile représentée par l'objet est vide, `False` sinon ;
- la méthode `empiler` qui prend en paramètre un élément et l'ajoute au sommet de la pile ;
- la méthode `depiler` qui renvoie la valeur du sommet de la pile et enlève cet élément.

10. Déterminer la valeur associée à la variable `elt` après l'exécution des instructions suivantes :

```
>>> essai = Pile()
>>> essai.empiler(3)
>>> essai.empiler(2)
>>> essai.empiler(10)
>>> elt = essai.depiler()
>>> elt = essai.depiler()
```

Lorsqu'il en existe un, un ordre de réalisation des tâches sera représenté par un objet de classe `Pile` contenant tous les sommets du graphe de manière à ce que les tâches qu'il faut réaliser en premier se retrouvent au sommet de la pile.

La fonction `ordre_realisation` est écrite de la manière suivante :

```
1  def ordre_realisation(graphe):
2      n = len(graphe)
3      ouverts = [ False for i in range(n) ]
4      fermes = [ False for i in range(n) ]
5      ordre = Pile()
6      ok = True
7      s = 0
8      while (ok and s < n):
9          ok = mystere(graphe, s, n, ouverts, fermes, ordre)
10          s = s + 1
11     if ok :
12         return ordre
13     return None
```

11. Sachant que dans la fonction `mystere`, la ligne 24 peut être remplacée par une ou plusieurs instructions, donner ce qu'il faut écrire pour que, lorsque c'est

possible, `ordre_realisation` renvoie effectivement un ordre de réalisation des tâches du graphe.