Partie 1

Scripts des algorithmes de parcours de graphes

1.1 Parcourir un graphe pour trouver TOUS les chemins

```python
def parcours(G, depart,lst_chemins, chemin = []):
    if chemin == []:
        chemin = [depart]
    for sommet in G[depart]:
        if sommet not in chemin:
            lst_chemins.append(chemin + [sommet])
            parcours(G, sommet, lst_chemins, chemin + [sommet])
    return lst_chemins
```

1.2 Parcours en largeur

```
VARIABLE
G : un graphe
s : noeud (origine)
u : noeud
v : noeud
f : file (initialement vide)

//On part du principe que pour tout sommet u du graphe G, u.couleur =
   blanc à l'origine
DEBUT
s.couleur ← rouge
enfiler (s,f)
tant que f non vide :
  u ← defiler(f)
  pour chaque sommet v adjacent au sommet u :
    si v.couleur n est pas rouge :
      v.couleur ← rouge
      enfiler(v,f)
    fin si
  fin pour
fin tant que
FIN
```

On donne l'implementation en python de l'algorithme BFS :

```python
from collections import deque

def bfs(graph, start):
    visited = []
    queue = deque()
    queue.append(start)
    while queue: # tq queue non vide
        node = queue.popleft()
        if node not in visited:
            visited.append(node)
            unvisited = [n for n in graph[node] if n not in visited]
```

```
            queue.extend(unvisited)
            #queue = queue + unvisited
    return visited
```

1.3 Parcours en profondeur

1.3.1 Recursif

```
VARIABLE
G : un graphe
u : noeud
v : noeud
//On part du principe que pour tout sommet u du graphe G, u.couleur = blanc à l'origine
DEBUT
PARCOURS_PROFONDEUR(G,u) :
  u.couleur ← rouge
  pour chaque sommet v adjacent au sommet u :
    si v.couleur n est pas rouge :
      PARCOURS_PROFONDEUR(G,v)
    fin si
  fin pour
FIN
```

1.3.2 itératif

```
VARIABLE
s : noeud (origine)
G : un graphe
u : noeud
v : noeud
p : pile (pile vide au départ)
//On part du principe que pour tout sommet u du graphe G, u.couleur = blanc à l'origine
DEBUT
s.couleur ← rouge
empiler(s,p)
tant que p n'est pas vide :
  u ← depiler(p)
  pour chaque sommet v adjacent au sommet u :
    si v.couleur n'est pas rouge :
      v.couleur ← rouge
      empiler(v,p)
    fin si
  fin pour
fin tant que
FIN
```

Partie 2

Exercices

2.1 méthodes de listes

Pour les exercices suivants, on definit 2 listes :

```
file = ['A','B','C']
unvisited = ['D','E','F']
```

On déroule un programme ligne après ligne. La liste `file` evolue au fur et à mesure.

Que vaut `file` après chacune des instructions :

```
> file = file.extend(unvisited)
> file.pop()
> file.pop(0)
> file.append('G')
> file = file + ['H']
> file.append(['I','J'])
```

2.2 Adapter un algorithme en python

Pour l'agorithme BFS : comment traduit-on en python :

1. `s.couleur  rouge`
2. `si v.couleur n est pas rouge :`
3. `v.couleur  rouge`

2.3 Comparer les algorithmes BFS et PARCOURS_PROFONDEUR

1. Quelle différence majeure voyez vous entre ces 2 algorithmes ?
2. Comment traduisez vous en français : `visited` et `unvisited` ?
3. Déterminer le parcours en largeur depuis le sommet `A` pour le graphe G1 suivant :

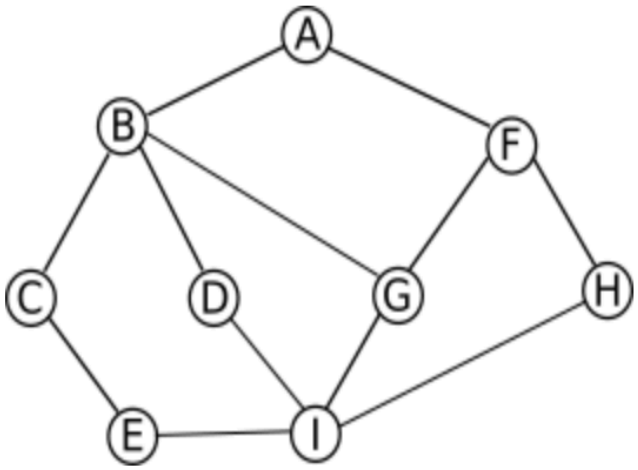

FIGURE 1 – graphe G1

4. Déterminer le parcours en profondeur depuis le sommet `A` pour le graphe G1

2.4 Parcourir selon les 3 algorithmes

On donne le dictionnaire de sommets voisins pour le graphe `G_mini` :

```
1 G_mini = {0: [1,3], 1: [0,2], 2: [1,3], 3:[1,3]}
```

1. Représenter ce graphe
2. Donner la liste retournée par la fonction `parcours` pour le graphe `G_mini`
3. Donner la liste retournée par la fonction `bfs` pour le graphe `G_mini`
4. Donner la liste retournée par la fonction `PARCOURS_PROFONDEUR` pour le graphe `G_mini`
5. Parmis les 3 listes, laquelle est un *chemin* ? Pourquoi ?
6. Pour la liste retournée par la fonction `parcours` : peut-il y avoir des doublons ? Pourquoi ?
7. Laquelle de ces 3 fonctions peut servir de base pour concevoir une fonction de detection de cycle dans le graphe ?