Implémentation et parcours de graphes

3h (avec documents)

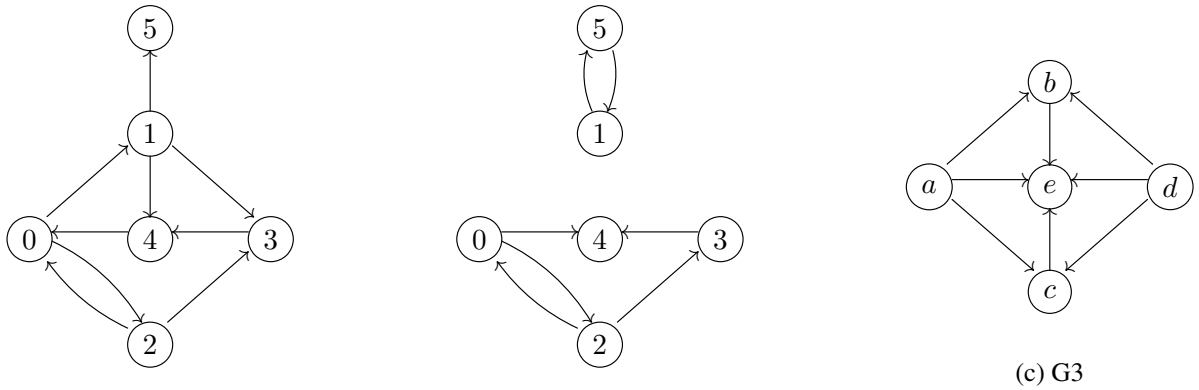
Année 2021-2022

Préambule

- Pour tester dans utop : **open** Be.Graphe;;
- Le code rendu doit **impérativement compiler**. Si une partie ne compile pas, mettez-la en commentaires, ça peut donner lieu à des points.
- Interdiction de modifier les signatures (nom + type) fournies. Les vérifications seront automatisées.
- Même si la non utilisation d'itérateur sera pénalisée, des points seront accordés si les fonctions sont écrites correctement sans itérateur.
- vous êtes autorisé à utiliser toutes les fonctions de la librairie standard, en particulier le module List : `https://ocaml.org/api/List.html`.
- L'exercice 3 peut nécessiter l'utilisation de l'exercice 2. L'exercice 6 peut nécessiter l'utilisation de l'exercice 5. **Les autres exercices sont indépendants.**

1 Introduction

Dans ce BE, nous allons uniquement utiliser la notion de graphe orienté. Un graphe est, de manière peu formelle, un ensemble de sommets reliés par des arêtes (orientées). Des exemples de graphes sont représentés dans la figure 3.



(a) G1 (b) G2 (c) G3

FIGURE 1 – Exemples de graphe

Les sommets accessibles directement à partir d'un sommet en n'empruntant qu'une seule arête s'appellent les **voisins**. Il peut y avoir plusieurs voisins si plusieurs arêtes partent d'un sommet. Par exemple, dans le graphe G1, les voisins de 0 sont 1 et 2, le voisin de 3 est 4 et 5 n'a pas de voisin.
Un sommet est **accessible depuis** un autre sommet, s'il existe une suite de voisins qui permet de passer de l'un à l'autre. Par exemple, dans le graphe G1, 3 est accessible depuis 0 (0 - 1 - 3) et 1 est accessible depuis 3 (3 - 4 - 0 - 1).

2 Première implémentation : liste de sommets et listes d'arêtes

Un graphe G est un couple (V,E) où V (vertices) est l'ensemble des sommets (nombre fini) et E (edges) l'ensemble des arêtes (couples de sommets de V). Par exemple, dans notre exemple G1,
V = {0,1,2,3,4,5} et E = {(0,1),(0,2),(1,3),(1,4),(1,5),(2,0),(2,3),(3,4),(4,0)}.
Un graphe peut être représenté en OCaml sous la forme :

```
type 'a graphe_ev = EV of 'a list * ('a * 'a) list
```

Dans les sources fournies g1_ev correspond au graphe de la figure 1a, g2_ev à celui de la figure 1b et g3_ev à celui de la figure 1c.

▷ **Exercice 1** *Une instance de type 'a graphe_ev n'est un graphe que si tous les sommets qui apparaissent dans la liste d'arêtes apparaissent également dans la liste de sommets.*

1. *Écrire le contrat d'une fonction* bienforme_ev *qui vérifie si une instance de type 'a graphe_ev est un graphe.*
2. *Écrire les tests unitaires de la fonction* bienforme_ev.
3. *Écrire la fonction* bienforme_ev.

▷ **Exercice 2** *Écrire la fonction* voisins_ev *qui renvoie la liste des voisins d'un sommet dans un graphe. Son contrat et ses tests unitaires sont donnés.*

▷ **Exercice 3** *Écrire la fonction* accessible_depuis_ev *qui renvoie la liste des sommets accessibles depuis un sommet dans un graphe. Son contrat et ses tests unitaires sont donnés.*
Aide : Vous pouvez utiliser une fonction auxiliaire qui aura deux paramètres : la liste des sommets à visiter et la liste des sommets déjà visités.

3 Seconde implémentation : liste de sommets

Une seconde représentation possible des graphes est une liste de couples (sommet, liste de voisins). Par exemple, dans notre exemple G1 = {(0,{1,2}),(1,{3,4,5}),(2,{0,3}),(3,{4}),(4,{0}), (5,{})}.
Un graphe peut être représenté en OCaml sous la forme :

```
type 'a graphe_s = Sommets of ('a * 'a list ) list
```

Dans les sources fournies g1_s correspond au graphe de la figure 1a, g2_s à celui de la figure 1b et g3_s à celui de la figure 1c.

▷ **Exercice 4** *Une instance de type 'a graphe_s n'est un graphe que si tous les sommets qui apparaissent dans les listes de voisins sont effectivement listés comme sommet.*

1. *Écrire le contrat d'une fonction* bienforme_s *qui vérifie si une instance de type 'a graphe_s est un graphe.*
2. *Écrire les tests unitaires de la fonction* bienforme_s.
3. *Écrire la fonction* bienforme_s.

▷ **Exercice 5** *Écrire la fonction* voisins_s *qui renvoie la liste des voisins d'un sommet dans un graphe. Son contrat et ses tests unitaires sont donnés.*

▷ **Exercice 6** *Écrire la fonction* accessible_depuis_s *qui renvoie la liste des sommets accessibles depuis un sommet dans un graphe. Son contrat et ses tests unitaires sont donnés.*
Aide : Vous pouvez utiliser une fonction auxiliaire qui aura deux paramètres : la liste des sommets à visiter et la liste des sommets déjà visités.

4 Conversion

4.1 Arbres → graphes

En supposant qu'il n'y ait pas deux valeurs identiques dans deux nœuds différents, les arbres binaires et n-aire vus en cours, TD et TP peuvent être vus comme des cas particuliers de graphe (graphes non cycliques).

▷ **Exercice 7**

1. *Écrire la fonction* arbre_binaire_to_graphe_ev *qui converti un arbre binaire en son graphe (première implémentation) associé. Contrairement aux arbres binaires, où le fait d'être un fils gauche ou droit peut être important, ici l'ordre des arêtes n'aura pas d'importance. Son contrat et ses tests unitaires sont donnés.*

FIGURE 2 – Arbre binaire → graphe

2. *Écrire la fonction* arbre_naire_to_graphe_ev *qui converti un arbre n-aire en son graphe (première implémentation) associé. Contrairement aux arbres n-aires, où l'ordre des fils peut être important, ici l'ordre des arêtes n'aura pas d'importance. Son contrat et ses tests unitaires sont donnés.*

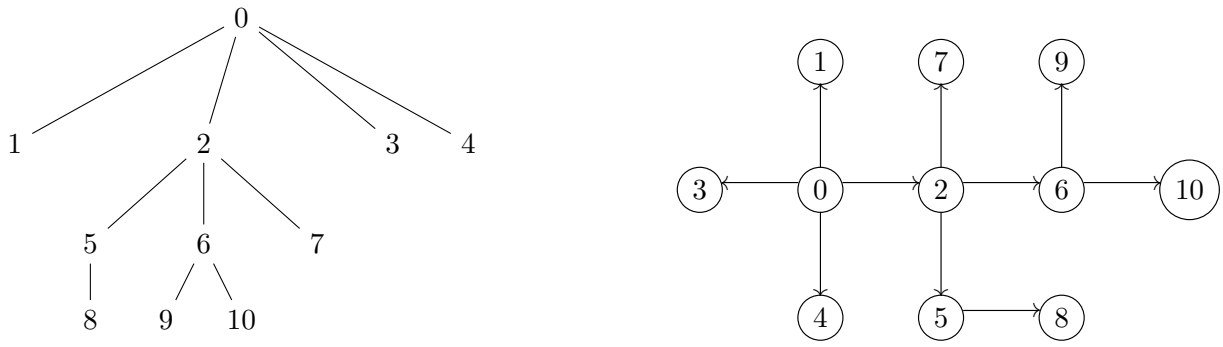


FIGURE 3 – Arbre n-aire → graphe

4.2 Graphes → graphes

▷ **Exercice 8**

1. *Écrire la fonction* graphe_ev_to_graphe_s *qui converti un graphe (première implémentation) en son graphe (seconde implémentation) associé. Son contrat et ses tests unitaires sont donnés.*
2. *Écrire la fonction* graphe_s_to_graphe_ev *qui converti un graphe (seconde implémentation) en son graphe (première implémentation) associé. Son contrat et ses tests unitaires sont donnés.*