

TP n°2 - Parcours de graphes

Notions abordées

- parcours préfixe d'arbre général en OCAML
- représentation de graphes orientés en OCAML
- parcours en profondeur récursif
- parcours en profondeur avec le module Stack
- parcours en largeur avec le module Queue
- différents mécanismes d'itération : fonctions récursives, `List.iter`, boucle `for`
- application de parcours pour différents problèmes sur un graphe de grande taille

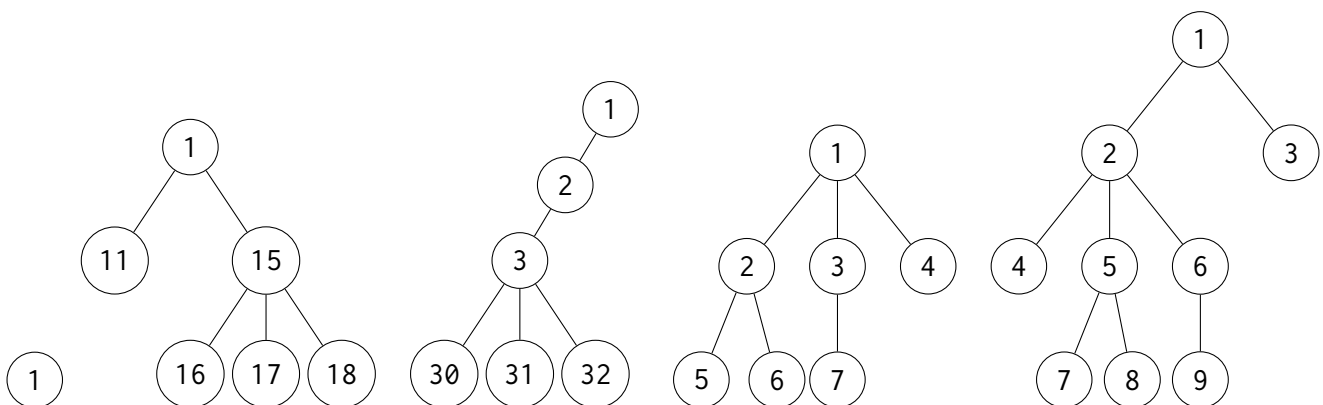
Exercice 1 : Parcours de graphes : implémentation pas à pas

1. Parcours d'arbre général

On fournit dans le fichier `compagnon_parcours.ml` le type ci-dessous pour représenter et les arbres généraux (*i.e.* dont les nœuds sont d'arité arbitrairement grande).

```
1 type 'a arbre_g =  
2   | Vide  
3   | Nd of 'a * ('a arbre_g) list
```

Afin de tester les fonctions à coder, le fichier `compagnon_parcours.ml` fournit aussi les définitions en OCAML des cinq arbres généraux représentés ci-dessous.



Q. 1 Définir une fonction `affiche_ag_prefixe` prenant en paramètre un arbre général étiqueté par des entiers et affichant les étiquettes de ses nœuds selon un parcours préfixe. Par exemple, pour l'arbre ci-dessus à droite le parcours est **1 2 4 5 7 8 6 9 3**.

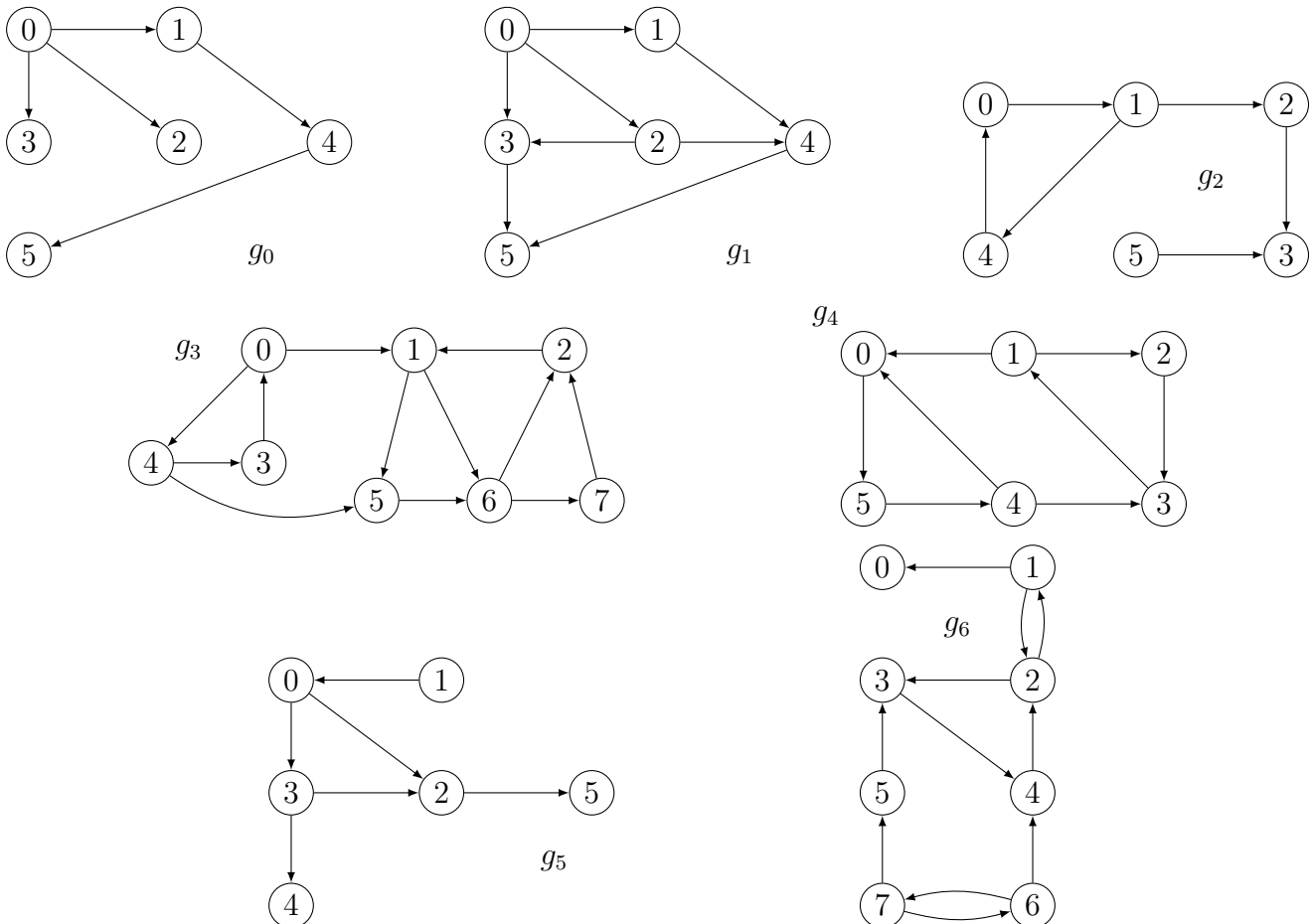
Cette fonction ne devra pas comporter deux fonctions mutuellement récursives, on utilisera la fonctionnelle `List.iter` du module `List` pour itérer sur la listes des enfants.

2. Parcours de graphe orienté

On s'intéresse maintenant au parcours de graphes orientés dont les sommets sont les entiers d'un intervalle $\llbracket 0, n - 1 \rrbracket$ avec $n \in \mathbb{N}$. On représente en machine de tels graphes par table de listes de successeurs. On se munit donc du type graphe ci-dessous pour l'implémentation en OCAML.

```
1 | type graphe = (* graphes par table de listes de successeurs *)
2 |   int list array
```

Afin de tester les fonctions à coder, le fichier `compagnon_parcours.ml` fournit aussi les définitions en OCAML des sept graphes représentés ci-dessous.



Dans toutes les fonctions à coder ici on évitera d'utiliser des fonctions mutuellement récursives. On utilisera la fonctionnelle `List.iter` du module `List` pour itérer sur les listes des successeurs.

Q. 2 Dans cette question on suppose que le graphe à parcourir est un arbre (*i.e.* le graphe non orienté obtenu en omettant l'orientation des arc est connexe et acyclique) et que tous les sommets sont accessibles depuis le sommet 0, comme le graphe g_0 par exemple.

Définir une fonction `imprime_parcours_profondeur_1` prenant en paramètre un tel graphe et affichant tous les sommets du graphe dans un ordre de parcours en profondeur.

Q. 3 Dans cette question on suppose que tous les sommets du graphe à parcourir sont accessibles depuis le sommet 0, comme les graphes g_1 et g_2 par exemple. **Attention on ne suppose plus que le graphe est un arbre.**

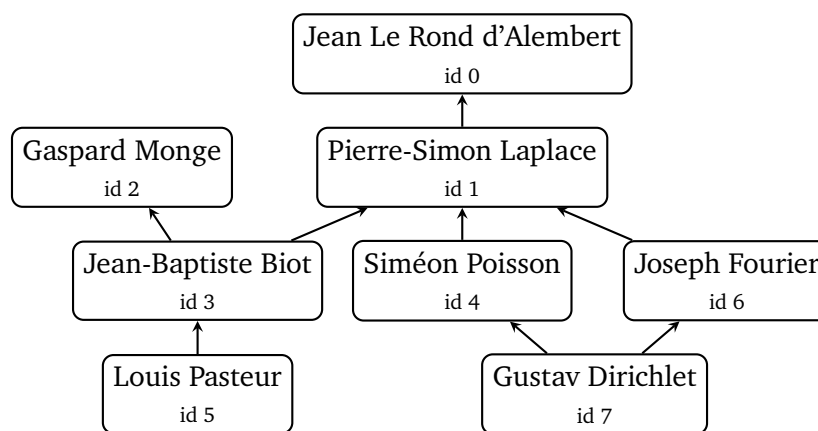
Définir une fonction `imprime_parcours_profondeur_2` prenant en paramètre un tel graphe et affichant tous les sommets du graphe dans un ordre de parcours en profondeur.

- Q. 4** Définir une fonction `imprime_parcours_profondeur_3` prenant en paramètre un graphe orienté quelconque et affichant tous les sommets du graphe dans un ordre de parcours en profondeur.
- Q. 5** Même question, dans une fonction `imprime_parcours_profondeur_4`, **mais on ne s'autorise plus le mécanisme d'itération récursif** pour l'exploration des descendants d'un sommet. Autrement dit, on n'utilise plus la pile des appels récursifs mais une pile explicite. On pourra utiliser l'implémentation de pile fournie par le module OCAML Stack.
- Q. 6** Définir une fonction `imprime_parcours_largeur`, qui prend en paramètre un graphe orienté quelconque et qui imprime les sommets du graphe dans l'ordre d'un **parcours en largeur**. On pourra utiliser l'implémentation de file fournie par le module OCAML Queue.

Exercice 2 : Généalogie mathématicienne

Le site <https://www.genealogy.math.ndsu.nodak.edu/> recense des mathématiciens et les thèses qu'ils ont soutenues, en vue d'obtenir un diplôme de doctorat. On apprend, par exemple, sur la page <https://www.genealogy.math.ndsu.nodak.edu/id.php?id=108295> que Laplace a soutenu sa thèse intitulée *Recherches sur le calcul intégral aux différences infiniment petites et aux différences finies* en 1769 à l'Université de Caen, en Normandie. La thèse de doctorat de Laplace a été encadrée par d'Alembert. À son tour, Laplace a été l'encadrant de deux thèses : celle de Biot en 1797 puis celle de Poisson en 1800. On peut d'ailleurs lire sur la page consacrée à Biot (<https://www.genealogy.math.ndsu.nodak.edu/id.php?id=294953>) que la thèse qu'il a soutenue en 1797, était encadrée par Laplace, mais aussi par Monge. Biot n'a encadré, quant à lui, qu'une seule thèse, celle de Pasteur, en 1847.

Dans cet exercice, on s'intéresse au graphe orienté dont les sommets sont les mathématiciens et dont les arcs représentent la relation "a été encadré par". On appelle ce graphe le **graphe d'encadrement**. On a représenté ci-dessous un extrait du graphe d'encadrement.



Représentation en OCAML. Afin de représenter en OCAML un tel graphe, on associe à chaque mathématicien un identifiant unique d'un intervalle entier $\llbracket 0, n-1 \rrbracket$, on représente alors le graphe au moyen d'une table de listes de successeurs : la case d'indice i contient la liste des encadrants du mathématicien d'identifiant i .

Le code `compagnon_mgp.ml` qui vous est fourni contient donc une définition d'un type graphe par table de listes de successeurs.

```
1 | type graphe = int list array (* Graphe par table de listes de successeurs *)
```

Afin de stocker les données associées à un mathématicien on se munit aussi d'un type structuré `mathematicien` donnant, son nom, l'année durant laquelle il a soutenu sa thèse, l'université et le pays dans lequel cette thèse a eu lieu.

```
1 | type mathematicien = {  
2 |   nom : string; (* Son nom *)  
3 |   annee : int option; (* L'année de sa thèse, si renseignée *)  
4 |   universite : string; (* L'université de sa thèse *)  
5 |   pays : string; (* Le pays de son université *)  
6 | }
```

On définit finalement le type `datas = mathematicien array` comme étant un tableau de taille n de mathématiciens, la case d'indice i d'un tel tableau contenant les données associées à la thèse du mathématicien d'identifiant i .

Une fois récupérés les fichiers "mgp_graph.txt" et "mgp_meta.txt", l'appel à la fonction fabrique : `string -> string -> graphe * datas` prenant en paramètres les chemins vers les fichiers, calcule le graphe des encadrements ainsi que les données des mathématiciens.

On propose ci-dessous un exemple de manipulation du graphe des encadrements.

```
# let g, d = fabrique "mgp_graph.txt" "mgp_meta.txt";
- : ...
# Array.find_index (fun x -> x.nom = "Jean-Baptiste Biot") d;;
- : int option = Some 286972
# d.(286972);;
- : mathematicien = {nom = "Jean-Baptiste Biot"; annee = Some 1797; universite = "École
↳ Polytechnique"; pays = "France"}
# g.(286972);;
- : int list = [105825; 286978]
# d.(105825);;
- : mathematicien = {nom = "Pierre-Simon Laplace"; annee = Some 1769; universite = "Université de
↳ Caen Normandie"; pays = "France"}
# d.(286978);;
- : mathematicien = {nom = "Gaspard Monge"; annee = None; universite = ""; pays = ""}
```

Afin de débbugger vos programmes sur un exemple simple, on vous fournit également un petit graphe `petit_graphe` : graphe et les données associées `petites_datas` : datas, correspondant exactement à l'exemple de la figure ci-dessus, avec les identifiants qui y sont indiqués (par exemple, `petites_datas.(1).nom = "Pierre-Simon Laplace"` et `petit_graphe.(3) = [1; 2]`, puisque Biot, d'identifiant 3, a été encadré par Laplace et Monge, d'identifiants 1 et 2).

1. Quelques statistiques

Dans les questions de cette section, il est demandé de donner des grandeurs de la base de données **Mathematics Genealogy Project**, on s'attend, bien sûr, à ce que vous trouviez ces grandeurs à l'aide d'un programme et non "à la main".

- Q. 1 Combien de mathématiciens se trouvent dans la base ?
- Q. 2 Donner les noms du ou des encadrants de la thèse de Sophie Germain.
- Q. 3 Combien d'arcs contient le graphe d'encadrement ?
- Q. 4 Donner la liste des identifiants des mathématiciens n'ayant aucun encadrant renseigné dans la base. Combien y en a-t-il ?
- Q. 5 Qui est la personne ayant eu le nombre maximal d'encadrants de thèses ? Et combien cette personne a-t-elle eu d'encadrants ?
- Q. 6 Combien de mathématiciens ont encadré au moins une thèse ?
- Q. 7 Qui est le mathématicien ayant encadré le nombre maximal de thèses ? Combien en a-t-il encadré ?

2. Parcours du graphe

Dans la suite de l'exercice, on dit qu'un mathématicien y est un **ancêtre** du mathématicien x dès lors qu'il existe un chemin de x à y dans le graphe d'encadrement. On appelle alors **distance d'encadrement de x à y** la distance, en nombre d'arcs, entre x et y .

- Q. 8 Implémenter une fonction `est_ancetre` : `graphe -> int -> int -> bool` prenant en paramètres un graphe `g` et deux identifiants de mathématiciens x et y et testant si y est un ancêtre de x (au sens de la relation encadrant).
Vérifier que :

- `(est_ancetre g 286972 105825)` est vrai (Laplace, identifiant 105825, est un ancêtre de Biot, identifiant 286972) ;
- `(est_ancetre g 104689 18104)` est vrai (Gauß, identifiant 18104, est un ancêtre de Gyula Vályi, identifiant 104689) ;
- `(est_ancetre g 105825 286972)`, `(est_ancetre g 317631 286972)` et `(est_ancetre g 317631 286972)` sont faux.

Q. 9 Implémenter une fonction `distance_ancetre : graphe -> int -> int -> int` option prenant en paramètres un graphe `g` et deux identifiants de mathématiciens `x` et `y` et retournant la distance d'encadrement de `x` à `y`. Si `y` n'est pas un ancêtre de `x`, la fonction retourne `None`. *Attention* : On n'utilisera pas la fonction `est_ancetre`.

Vérifier que les résultats coïncident avec ceux de la question précédente, en sachant que Biot a été directement encadré par Laplace et que Gyula Vályi a pour ancêtre Gauß, à distance 6.

Q. 10 Définir une fonction `graphe_miroir : graphe -> graphe` prenant en paramètres un graphe, représenté par table de listes d'adjacence, et retournant son graphe miroir, représenté par table de listes d'adjacence. Le graphe miroir d'un graphe orienté $G = (S, A)$ est le graphe $G' = (S, \{(y, x) \mid (x, y) \in A\})$.

Q. 11 Définir une fonction `nb_descendants : graphe -> int -> int` prenant en paramètres un graphe d'encadrement et l'identifiant d'un mathématicien et retournant le nombre de mathématiciens dont il est l'ancêtre (son nombre de descendants donc).

On dit que deux mathématiciens `x` et `y` **sont liés** dès lors qu'il existe $n \in \mathbb{N}$ et une chaîne de mathématiciens $x = m_0, m_1, m_2, \dots, m_n = y$ tels que pour tout $i \in \llbracket 1, n \rrbracket$ m_{i-1} est un encadrant de m_i ou m_i est encadrant de m_{i-1} .

Q. 12 Donner le nombre de classes d'équivalence pour la relation "être lié". Donner, sous forme de liste de couples, l'histogramme des nombres de classes d'équivalence en fonction des effectifs desdites classes. Par exemple `[(152, 3); (13, 2); (4, 5)]` indiquerait que le graphe contient 3 classes contenant 152 mathématiciens, 13 classes contenant 2 mathématiciens et 4 classes contenant 5 mathématiciens.

Vérifier que l'on trouve 15129 classes au total, dont une classe de 309968 mathématiciens (soit le couple `(309968, 1)` dans l'histogramme).

Q. 13 Étonnamment, le graphe d'encadrement de **Mathematics Genealogy Project** contient un circuit. Trouver ce circuit, supprimer un des arcs du circuit, vérifier qu'il n'y a alors plus de circuit.

On suppose dorénavant que le graphe est sans circuit. On appelle **lignée** un chemin dans le graphe d'encadrement.

Q. 14 Trouver la plus longue lignée de mathématiciens se trouvant dans la base de données.

Vérifier que l'on trouve une lignée de 62 mathématiciens, du mathématicien d'identifiant 307544 (Abu Mansur al-Hasan ibn Nuh al-Qumri) au mathématicien d'identifiant 276663 (Jinwei Ye).

Exercice 3 : Un cas pratique

Dans cet exercice, on s'intéresse à la situation concrète suivante. On organise une sortie scolaire dont le retour s'effectue en car. L'heure de retour étant incertaine, on souhaite mettre en place un protocole d'appels téléphoniques afin de prévenir toutes les familles le moment venu. L'idée est d'avoir à appeler seulement quelques personnes, qui elles-mêmes appelleront d'autres personnes et ainsi de suite. On suppose qu'on connaît les liens d'amitié entre les familles, ainsi on choisit de ne demander à une famille d'appeler que des familles amies. Enfin, afin que ce protocole soit clair, on souhaite qu'il soit formulé par une liste de qui appelle qui (on suppose qu'on connaît le nom de chaque famille, et que ces noms sont deux à deux distincts). Par exemple :

Les profs appellent les Dupont
Les Dupont appellent les Durand
Les Dupond appellent les Dupont

- Q. 1** Proposer en C une structure permettant de représenter les données en entrées.
- Q. 2** Comment calculer un protocole d'appels valide, c'est-à-dire permettant d'informer toutes les familles, et minimal, c'est-à-dire dans lequel l'équipe organisatrice a un nombre minimal d'appels à passer ? Expliquer cette démarche à l'oral en faisant le lien entre les notions vues en cours et cette situation concrète.
- Q. 3** Proposer en C une fonction `afficher_protocole` qui affiche un protocole d'appels valide et minimal.