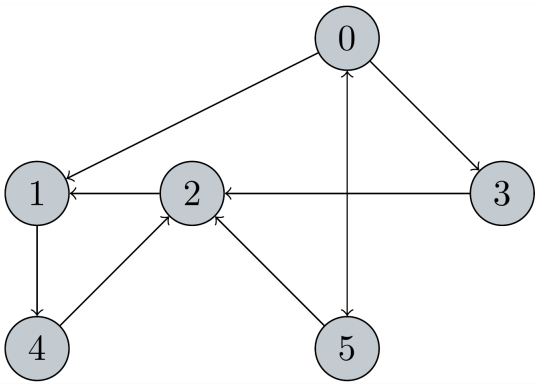
TP n°06 – Parcours en profondeur et deux applications

On considère dans ce TP des graphes $G = (S, A)$ où S est $[\![0; n-1]\!]$ avec n le nombre de sommets. Les graphes sont représentés par le type Ocaml `int list array` : un tableau t de taille n tel que $t.(i)$ contient la liste des voisins de i, dans un ordre quelconque.

- **Q1.** Définir ce graphe orienté en Ocaml :



- **Q2.** Définir également le graphe où les flèches deviennent des arêtes non-orientées.

1 Le parcours en profondeur

- **Q3.** Écrire une fonction `parcours_profondeur : int list array -> int -> unit` réalisant le parcours en profondeur à partir d'un sommet s et affichant les sommets dans l'ordre où ils sont parcourus.
- **Q4.** Écrire une fonction `parcours_profondeur_complet : int list array -> unit` qui réalise le parcours en profondeur complet du graphe et affiche les sommets dans l'ordre de leur parcours.
- **Q5.** Tester sur les exemples de graphes des questions 1 et 2.

2 Bicolorabilité

Le problème du k-coloriage consiste à associer à chaque sommet d'un graphe une étiquette (ou couleur) dans $[\![0, k-1]\!]$, telle que deux noeuds voisins n'ont pas la même couleur. Ici on considèrera le cas où k vaut 2, et les graphes sont non-orientés.

- **Q6.** Dessiner un graphe bi-coloriable non-orienté à $n = 6$ sommets.
- **Q7.** Supposons qu'un graphe $G = (S, A)$ soit bi-coloriable. Comment peut-on trouver un coloriage à deux couleurs qui marche ? Indice : utiliser le parcours en profondeur.
- **Q8.** Comment adapter l'algorithme précédent pour déterminer si un graphe est bi-coloriable, et renvoyer un coloriage s'il l'est ?
- **Q9.** Montrer qu'un cycle de taille impair n'est pas bicoloriable.

On veut montrer qu'un graphe G qui n'admet pas de cycle impair est bicoloriable. On se place sans perte de généralité dans un graphe G connexe et on fixe un sommet quelconque s.

- **Q10.** Pourquoi est-ce sans perte de généralité ?
- **Q11.** Montrer que si le plus court chemin entre s et un sommet u est de taille $2 * a$ (resp. $2 * a + 1$) et le plus court chemin entre s et un sommet v est de taille $2 * b$ (resp. $2 * b + 1$), alors il ne peut pas exister d'arrête entre u et v.
- **Q12.** Conclure que G est bi-coloriable. Indice : l'algorithme de la question 7 fait partie de la justification.
- **Q13.** Montrer qu'un graphe est bi-coloriable si et seulement si il n'admet pas de cycle de longueur impaire.
- **Q14.** En déduire une preuve de correction pour votre algorithme de la question 8.
- **Q15.** Implémenter une fonction `bicoloriage : int list array -> int array option` qui prend en entrée un graphe et renvoie un coloriage si le graphe est coloriable et `None` sinon.

3 Les cycles eulériens

On considère ici $G = (S, A)$ non-orienté connexe. Un **cycle eulérien** est un cycle qui passe une et une seule fois par chaque arête du graphe.

- **Q16.** Montrer que si G admet un cycle eulérien alors les degrés de tous ses sommets sont pairs.
- **Q17.** Montrer l'inverse : si G est un graphe dont le degré de tous les sommets est pair, alors il admet un cycle eulérien. Indication : raisonner par récurrence forte sur le nombre d'arêtes.

- **Q18.** En déduire un algorithme simple permettant de savoir si un graphe admet un cycle eulérien. Cet algorithme permet-il de construire un cycle eulérien ?
- **Q19.** En adaptant un algorithme de parcours, écrire une fonction `construit_cycle : int list array -> int list` qui renvoie un cycle eulérien, sous la forme d'un tableau indiquant pour chaque sommet lequel le suit dans le cycle.