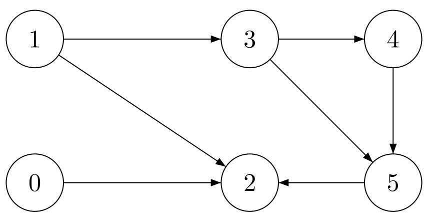
Colle 18 : Graphes orientés et ordonnancement

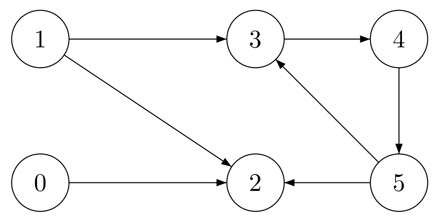
le 13 mai

1 Graphe orienté

Un graphe orienté est un couple $G = (S, A)$ ou S est l'ensemble des sommets de G et $A \subset S \times S$ est l'ensemble des arêtes (orientées) du graphe. Voici deux exemples de graphes orientés : le premier sans cycle, le second avec un cycle.



g1 : graphe orienté sans cycle



g2 : graphe orienté avec cycle

Les sommets sont numérotés de 0 à $|S| - 1$ et les graphes seront implémentés par des tableaux de listes d'adjacences :

```
type graphe = int list array;;
let (g1:graphe) = [|[2];[2;3];[];[4;5];[5];[2]|];;
let (g2:graphe) = [|[2];[2;3];[];[4];[5];[2;3]|];;
```

1.1 Quelques fonctions utiles

les fonctions de cette partie seront utilisées dans la partie suivante.

▷ **Question 1.** Ecrire une fonction `cree_tab_entrant : graphe -> int array` de complexité $O(|A|)$ telle que si g représente le graphe (S, A) alors `cree_tab_entrant g` renvoie un tableau `deg` donnant, pour chaque sommet de S le nombre d'arêtes aboutissant à ce sommet.
Exemples :

```
cree_tab_entrant g1 -> [|0;0;3;1;1;2|]
cree_tab_entrant g2 -> [|0;0;3;2;1;1|]
```

◁

▷ **Question 2.** Ecrire une fonction `sans_entrant : int array -> int list` telle que `sans_entrant deg` renvoie la liste des sommets auxquels aucune arête n'aboutit. La complexité doit être en $O(|S|)$.
Exemple :

```
sans_entrant (cree_tab_entrant g1) -> [0;1]
```

◁

▷ **Question 3.** Ecrire une fonction `est_vecteur_nul : int -> bool` qui détermine si toutes les cases d'un tableau passé en entrée sont nulles. ◁

▷ **Question 4.** Ecrire une fonction récursive `sup_entrants : int array-> int list -> int list-> int list` telle que `sup_entrants del l le` diminue le degré des sommets de `le` de 1 et renvoie la liste obtenue en ajoutant à `l` les sommets de `le` dont le degré est alors devenu nul. On suppose que la liste `le` ne contient que des sommets de degrés initialement non-nuls.

Exemple :

```
let de = [|0;1;2;2;1;0;1;0;0;2|];;
sup_entrants de [0;5] [1;2;4;9] -> [4;1;0;5]
```

◁

1.2 Graphes sans cycle

Nous allons ici utiliser les fonctions de la partie précédente.

Nous admettrons les deux propriétés suivantes :

- Un graphe dont l'ensemble des arêtes n'est pas vide et n'ayant aucun sommet de degré entrant nul (sauf éventuellement les sommets isolés) contient au moins un cycle.
- Soit g un graphe; s un sommet de g de degré entrant nul et g' le graphe obtenu en supprimant de g toutes les arêtes d'origine s. Alors g est sans cycle si et seulement si g' est sans cycle.

Par ailleurs, il est clair qu'un graphe dont l'ensemble des arêtes est vide est sans cycle.

Algorithme : Soit g un graphe. Par suppression successive des arêtes issues d'un sommet de degré entrant nul, on obtient un graphe g' n'ayant plus de sommet de degré entrant nul. Le graphe g est sans cycle si et seulement si le graphe ainsi obtenu est sans arête.

Implémentation efficace Pour une implémentation efficace, on utilisera uniquement le tableau deg_e et une liste des sommets de degré entrant nul. Ces deux données seront mises à jour en utilisant la fonction sup_entrants.

▷ **Question 5.** Ecrire une fonction `est_ss_cycle : graphe -> bool` telle que `est_ss_cycle g` renvoie `true` si et seulement si `g` est l'implémentation d'un graphe sans cycle. La complexité de cette fonction doit être en $O(|S| + |A|)$. ◁

Dans la suite de cette partie, on suppose que les graphes donnés en argument sont toujours sans cycles : cette condition n'a pas à être vérifiée par les fonctions demandées

Une relation d'ordre total est représentée par la liste en ordre croissant de ses éléments; ainsi la relation d'ordre $\prec$ sur $\{0, 1, 2, 3, 4, 5\}$ telle que $3 \prec 1 \prec 2 \prec 5 \prec 0 \prec 4$ est représentée par la liste $[3; 1; 2; 5; 0; 4]$.

Soit $G = (S, A)$ un graphe orienté sans cycle et $\prec$ une relation d'ordre total sur S. La relation $\prec$ est dite compatible avec le graphe G si et seulement si pour tout $(i, j) \in A$, on a $i \prec j$.

Nous admettrons que pour tout graphe G orienté sans cycle, il existe des relations d'ordre total compatibles avec G.

Soit g un graphe sans cycles, une liste $[a_1, a_2, \ldots, a_n]$ représente un ordre compatible avec g si et seulement si a_1 est un sommet de degré entrant nul et $[a_2, \ldots, a_n]$ est compatible avec le graphe obtenu en supprimant les arêtes issues de a_1. On peut ainsi implémenter un algorithme créant un ordre compatible avec un graphe g (ou testant si un ordre donné est compatible).

Comme pour l'algorithme précédent, on utilisera le tableau `deg` et une liste des sommets de degré entrant nul. Ces deux données seront mises à jour en utilisant la fonction `sup_entrants`.

▷ **Question 6.** Ecrire une fonction `un_ordre : graphe -> int list` telle que `un_ordre g` renvoie la liste représentant un ordre compatible avec le graphe sans cycle g.

Par exemple `un_ordre g1` peut renvoyer `[0 ; 1 ; 3 ; 4 ; 5 ; 2]` (ou n'importe quel autre ordre compatible).

La complexité de cette fonction doit être en $O(|S| + |A|)$. ◁

▷ **Question 7.** Ecrire une fonction `ordre_compatible : graphe -> int list -> bool` telle que `ordre_compatible g l` renvoie `true` si et seulement si `l` est la liste représentant un ordre compatible avec le graphe associé à g. Par exemple :

```
ordre_compatible g1 [0;1;3;4;5;2] -> true
ordre_compatible g1 [1;3;0;4;5;2] -> true
ordre_compatible g1 [1;2;0;4;2;5] -> false
```

◁

▷ **Question 8.** Ecrire une fonction récursive `somme_l : int list -> int` telle que `somme_l l` renvoie la somme des éléments de la liste `l`.

Ecrire une fonction récursive `nb_chemins : graphe -> int -> int -> int` telle que `nb_chemins g i j` renvoie le nombre de chemins menant du sommet `i` au sommet `j` dans le graphe sans cycle `g`. On pourra utiliser la fonction `List.map`. La terminaison de la fonction sera assurée par l'absence de cycle dans le graphe.

Ecrire, sur le modèle de la stratégie ci dessus, une fonction `max_chemin : graphe -> int ->int -> int` qui renvoie la longueur d'un plus long chemin de `i` à `j` dans le graphe.

En déduire une fonction `max_chemin_depuis : graphe -> int -> int` telle que `max_chemin_depuis g i` renvoie la longueur d'un plus long chemin d'origine `i` dans `g`.

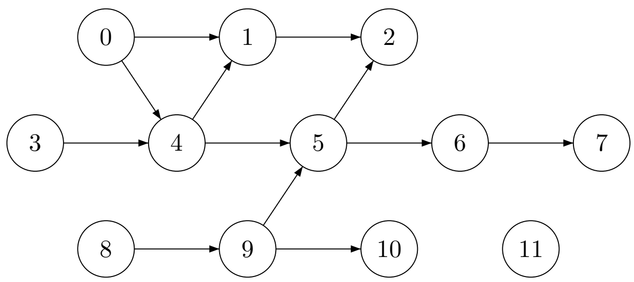
Améliorer la fonction précédente sans considérer toutes les destinations possibles mais uniquement celles obtenues avec un parcours en profondeur. ◁

2 Ordonnancement de tâches en parallèles

On dispose de m processeurs pour effectuer un ensemble de n tâches ; chaque tâche s'exécute en une unité de temps. Par ailleurs, un ordonnancement, donné par un graphe orienté acyclique, doit être respecté : autrement dit la tâche i ne peut être exécutée que si toutes les tâches qui la précèdent dans l'ordonnancement ont été exécutées

On appelle niveau d'une tâche i, la longueur du plus long chemin commençant en i, dans le graphe d'ordonnancement. Par exemple, pour l'ordonnancement suivant :

graghe g3



les niveaux des tâches sont :

tâche	0	1	2	3	4	5	6	7	8	9	10	11
niveau	4	1	0	4	3	2	1	0	4	3	0	0

Algorithme de Hu On effectue les tâches disponibles par ordre de niveau décroissant. Cet algorithme donne une exécution en temps minimal lorsque, pour chaque tâche u , il existe au plus une tâche v telle que $u \prec v$. Il peut bien sûr être utilisée même s'il existe plusieurs tâches v telles que $u \prec v$, mais l'exécution ne sera pas toujours en temps minimal

Pour l'exemple précédent, cet algorithme donne, lorsque l'on dispose de 3 processeurs P1, P2 et P3 :

tâches disponibles	P1	P2	P3
(0,3,8,11)	0	3	8
(4,9,11)	4	9	11
(1,5,10)	5	1	10
(2,6)	6	2	_
(7)	7	_	_

▷ **Question 9.** Ecrire une fonction `cree_niveau : graphe -> graphe` telle que `cree_niveau` renvoie le tableau des niveaux du graphe `g`. ◁

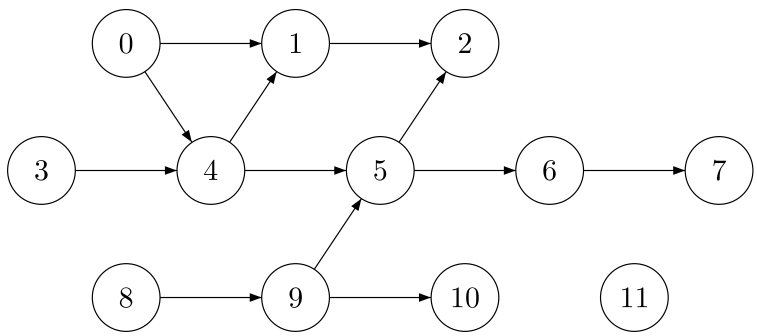
▷ **Question 10.** Ecrire une fonction `insere : 'a array -> int -> int list -> int list` telle que `insere niv i l` renvoie la liste l' obtenue en insérant `i` dans la liste `l` de sorte que l' soit classée par ordre décroissant de niveau. ◁

▷ **Question 11.** Ecrire une fonction `sans_entrant_classe : 'a array -> int array -> int list` telle que `sans_entrant_class niv deg` renvoie la liste des sommets auxquels aucune arête n'aboutit, classée par ordre décroissant de niveau.
◁

▷ **Question 12.** Ecrire une fonction `hu : graphe -> int -> int array array` telle que `hu g m` renvoie la matrice réalisant l'algorithme de Hu pour m processeurs. On écrira plusieurs fonctions courtes en s'inspirant de la première partie du TD. ◁

Annexe un exemple pour lequel l'algorithme de Hu ne donne une solution optimale

graghe g3



les niveaux des tâches sont:

tâche	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
niveau	4	4	4	4	4	4	3	2	1	0	3	2	1	0	0	0	0	0	0	0	0

L'algorithme de Hu donne, pour $m = 3$:

tâches disponibles	P1	P2	P3
$(0, 1, 2, 3, 4, 5, 10)$	0	1	2
$(3, 4, 5, 10)$	3	4	5
$(6, 10)$	6	10	–
$(7, 11)$	7	11	–
$(8, 12)$	8	12	–
$(9, 13, 14, 15)$	9	13	14
(15)	15	16	17
(15)	18	19	20

Cette exécution n'est pas optimale comme le montre l'exécution suivante :

tâches disponibles	P1	P2	P3
$(0, 1, 2, 3, 4, 5, 10)$	0	1	10
$(2, 3, 4, 5, 11)$	2	3	11
$(4, 5, 12)$	4	5	12
$(6, 13, 14, 15, 16)$	6	13	14
$(7, 16, 17)$	7	15	16
$(8, 13, 14, 15, 16)$	8	17	18
$(9, 15, 16)$	9	19	20