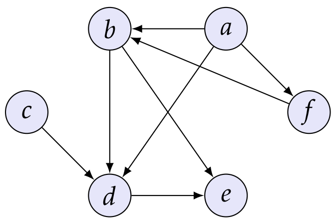
Graphes ordonnés

Durée : 2 heures

Dans tout le problème, on considère un graphe non orienté à $n \geqslant 1$ sommets $G = (S, A)$ où S désigne l'ensemble des sommets de G et A l'ensemble de ses arêtes.
Dans la première partie de ce contrôle, les graphes seront représentés en Python par un dictionnaire de *listes d'adjacence* (attention Valentin !) à l'instar du graphe G_1 représenté figure 1.




```
G1 = {'a': ['b', 'd', 'f'],
      'b': ['d', 'e'],
      'c': ['d'],
      'd': ['e'],
      'e': [],
      'f': ['b']}
```

Figure 1 – Le graphe G_1 et sa représentation en Python.

On appelle *cycle* une suite finie de sommets $(s_1, s_2, \ldots, s_k) \in S^k$ avec $k \geqslant 3$ telle que :

(i) $\forall i \in [\![1, k-1]\!], (s_i, s_{i+1}) \in A$;

(ii) $s_k = s_1$.

Un graphe dans lequel il n'existe pas de cycle est dit *acyclique*.

Partie I. Tri topologique

On dit que le graphe G peut être *ordonné* lorsqu'il existe une permutation $(s_0, \ldots, s_{n-1})$ de ses sommets de sorte que pour toute arête $(s_i, s_j) \in A$ on a $i < j$. On dit alors que la permutation $(s_0, \ldots, s_{n-1})$ est un *tri topologique* des sommets de G (il n'y a pas nécessairement unicité du tri topologique). Par exemple, le graphe G_2 représenté figure 2 peut être ordonné, et (d, c, e, b, f, a) est un tri topologique de ses sommets.

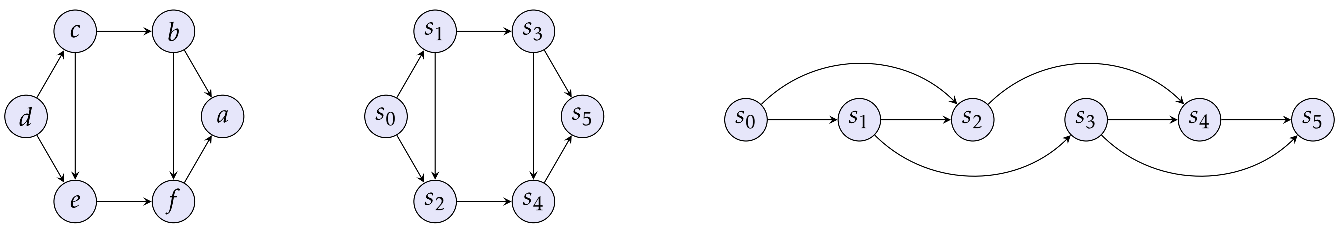


Figure 2 – Un exemple de graphe ordonné. On notera l'existence d'une représentation linéaire de ce graphe pour laquelle les arcs sont tous orientés de gauche à droite.

Question 1. Montrer qu'un graphe ordonné est acyclique.

Question 2. Montrer que le graphe G_1 peut être ordonné, et donner un tri topologique de ses sommets.

Question 3. Montrer que dans un graphe acyclique il existe au moins un sommet sans successeur.

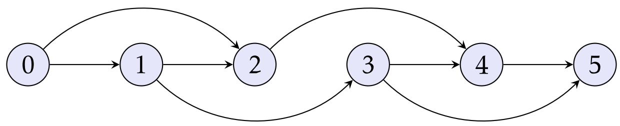
Question 4. En raisonnant par récurrence sur $n \in \mathbb{N}^*$ montrer que dans tout graphe acyclique il existe une permutation des sommets qui ordonne G. *On pourra considérer un sommet sans successeur et le graphe G' obtenu en supprimant ce sommet et toutes les arêtes qui y mènent.*

Question 5. Rédiger en Python une fonction `supprime(G, s)` qui prend pour argument un graphe G représenté par un dictionnaire de listes d'adjacence (attention Valentin !) et un sommet s et qui renvoie un *nouveau* graphe G' obtenu en supprimant s et toutes les arêtes qui y démarrent ou y aboutissent.

Question 6. En déduire une fonction `ordonne(G)` qui prend pour argument un graphe acyclique et qui renvoie un tri topologique de ses sommets (sous forme de liste). Par exemple, appliqué au graphe G_2 de la figure 2 cette fonction renverra la liste $[d, c, e, b, f, a]$ (ou tout autre liste ordonnant G_2).

Partie II. Plus long chemin dans un graphe acyclique

Dans cette partie, on considère un graphe acyclique $G = (S, A)$ dans lequel les sommets ont été remplacé par leur ordre topologique. Ainsi, le graphe G_2 de la figure 2 est maintenant dessiné et représenté en Python comme indiqué figure 3.




```
G2 = [[1, 2], [2, 3], [4], [4, 5], [5], []]
```

Figure 3 – le graphe ordonné G_2 et sa représentation en Python

Le problème qui nous intéresse maintenant est de déterminer la longueur maximale d'un (éventuel) chemin menant du sommet 0 au sommet $n-1$.

Question 7. Montrer que si tous les sommets à l'exception du sommet $n-1$ ont au moins un successeur, il existe au moins un chemin menant du sommet 0 au sommet $n-1$. *Dans la suite du problème, on supposera cette hypothèse toujours vérifiée.*

Un algorithme glouton

On considère l'algorithme glouton suivant : *partant du sommet 0, on suit le chemin d'un sommet s à son successeur de valeur minimale, jusqu'à aboutir au sommet $n-1$.*

Question 8. Justifier que cet algorithme ce termine mais qu'il ne donne pas systématiquement un chemin de longueur maximale (on construira un contre-exemple).

Question 9. Rédiger en Python une fonction `glouton(G)` qui, en suivant cet algorithme, renvoie la longueur d'un chemin reliant le sommet 0 au sommet $n-1$.

Programmation dynamique et mémoïsation

Pour tout $k \in [\![0, n-1]\!]$, on note $\ell(k)$ la longueur maximale d'un chemin reliant le sommet k au sommet $n-1$.

Question 10. Pour tout $k \leqslant n-2$, exprimer $\ell(k)$ en fonction des valeurs $\ell(k+1), \dots, \ell(n-1)$.

Question 11. En déduire une fonction *récursive* `longeurMax(G, k)` utilisant un dictionnaire mémorisant les calculs intermédiaires, qui prend pour argument un graphe ordonné G et un sommet k et renvoie la longueur du plus long chemin reliant les sommets k et $n-1$.

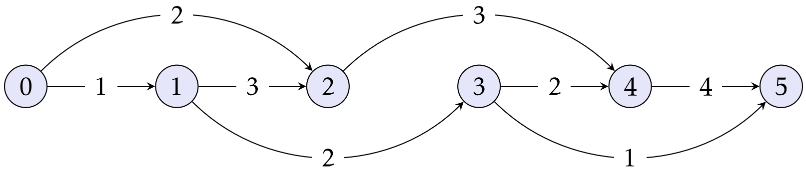
Plus long chemin

Question 12. Rédiger cette fois une fonction *non récursive* `longueursMax(G)` qui prend pour argument un graphe ordonné et renvoie la liste $[\ell(0), \ell(1), \dots, \ell(n-1)]$ (on s'interdira bien entendu d'utiliser la fonction définie à la question précédente).

Question 13. Rédiger enfin une fonction `cheminMax(G)` qui, en utilisant le tableau construit à la question précédente, renvoie un chemin de longueur maximale reliant le sommet 0 au sommet $n-1$.

Partie III. Chemin de poids maximal dans un graphe pondéré

Dans cette partie, on considère un graphe ordonné G dont les arêtes $(i, j) \in A$ sont pondérées par une valeur $g(i, j) \in \mathbb{R}$. On convient alors de poser $g(i, j) = -\infty$ lorsque $(i, j) \notin A$, ce qui permet de représenter G en Python par une matrice d'adjacence (attention Valentin !) (illustration figure 4).
L'objectif est ici de déterminer le poids maximal d'un chemin reliant le sommet 0 au sommet $n-1$.




```
G3 = [[-inf,    1,    2, -inf, -inf, -inf],
      [-inf, -inf,    3,    2, -inf, -inf],
      [-inf, -inf, -inf, -inf,    3, -inf],
      [-inf, -inf, -inf, -inf,    2,    1],
      [-inf, -inf, -inf, -inf, -inf,    4],
      [-inf, -inf, -inf, -inf, -inf, -inf]]
```

FIGURE 4 – un graphe ordonné muni d'une pondération, et sa représentation en Python

Vous pourrez librement utiliser une variable globale `inf` représentant le flottant $+\infty$.

Question 14. Pour tout $k \in [\![0, n-1]\!]$ on note $p_m(k)$ le poids maximal d'un chemin reliant le sommet k au sommet $n-1$. Rédiger une fonction `poidsMax(G)` prenant pour argument un graphe pondéré ordonné et renvoyant le poids maximal $p_m(0)$ d'un chemin reliant le sommet 0 au sommet $n-1$. Quelle est sa complexité temporelle ?

Pour finir, on s'impose une contrainte supplémentaire : relier le sommet 0 au sommet $n-1$ par un chemin de poids maximal *comprenant exactement r arêtes* (avec $r \in [\![1, n-1]\!]$ fixé).

Question 15. Pour tout $k \in [\![0, n-1]\!]$, pour tout $r \in [\![1, n-1]\!]$, on note $\mathrm{M}(k, r)$ le poids maximal d'un chemin de longueur k reliant le sommet k au sommet $n-1$, avec la convention $\mathrm{M}(k, r) = -\infty$ s'il n'existe pas de tel chemin.

a) Que valent respectivement $\mathrm{M}(n-1, r)$ et $\mathrm{M}(k, 0)$?

b) Pour $k \in [\![0, n-2]\!]$ et $r \in [\![1, n-1]\!]$, exprimer $\mathrm{M}(k, r)$ en fonction des $\mathrm{M}(i, s)$ avec $i > k$ et $s < r$.

c) en déduire en Python une fonction `poidsMaximum(G)` qui prend pour argument un graphe pondéré ordonné et qui renvoie le tableau des $\mathrm{M}(k, r)$ pour $0 \leqslant k \leqslant n-1$ et $0 \leqslant r \leqslant n-1$.